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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import RandomOverSampler

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('weatherAUS.csv')
df.head()
```

Out[2]:

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

5 rows × 23 columns

In [3]:
```
df.shape
```

Out[3]:
```
(8425, 23)
```

In [4]:
```
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
```

In [5]:
```
df.describe()
```

Out[5]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9
count	8350.000000	8365.000000	8185.000000	4913.000000	4431.000000	7434.000000	8349.000
mean	13.193305	23.859976	2.805913	5.389395	7.632205	40.174469	13.847
std	5.403596	6.136408	10.459379	5.044484	3.896235	14.665721	10.174
min	-2.000000	8.200000	0.000000	0.000000	0.000000	7.000000	0.000
25%	9.200000	19.300000	0.000000	2.600000	4.750000	30.000000	6.000
50%	13.300000	23.300000	0.000000	4.600000	8.700000	39.000000	13.000
75%	17.400000	28.000000	1.000000	7.000000	10.700000	50.000000	20.000
max	28.500000	45.500000	371.000000	145.000000	13.900000	107.000000	63.000

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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 [7]:
```
df.columns
```

Out[7]:
```
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')
```

In [8]:
```
df.rename(str.strip,
          axis='columns',
          inplace=True)

df.columns
```

Out[8]:
```
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')
```

In [11]:
```
categorical_features = [column_name for column_name in df.columns if df[column_name].d
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', 'WindDir3p
m', 'RainToday', 'RainTomorrow']
```

In [12]:
```
numerical_features = [column_name for column_name in df.columns if df[column_name].dty
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', 'W
indGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressur
e9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']
```

In [13]:
```
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
```

```
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
```

In [14]:
```
df['Date'] = pd.to_datetime(df['Date'])
df['year'] = df['Date'].dt.year
df['month'] = df['Date'].dt.month
df['day'] = df['Date'].dt.day
```

In [15]:
```
df.drop('Date', axis = 1, inplace = True)
df.head()
```

Out[15]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Wi
0	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	
1	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	
2	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	
3	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	
4	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	

5 rows × 25 columns

In [17]:
```
categorical_features = [column_name for column_name in df.columns if df[column_name].d
df[categorical_features].isnull().sum()
```

Out[17]:
```
Location          0
WindGustDir     991
WindDir9am      829
WindDir3pm      308
RainToday       240
RainTomorrow    239
dtype: int64
```

In [18]:
```
categorical_features_with_null = [feature for feature in categorical_features if df[fe
for each_feature in categorical_features_with_null:
    mode_val = df[each_feature].mode()[0]
    df[each_feature].fillna(mode_val,inplace=True)
```

In [19]:
```
numerical_features = [column_name for column_name in df.columns if df[column_name].dty
df[numerical_features].isnull().sum()
```

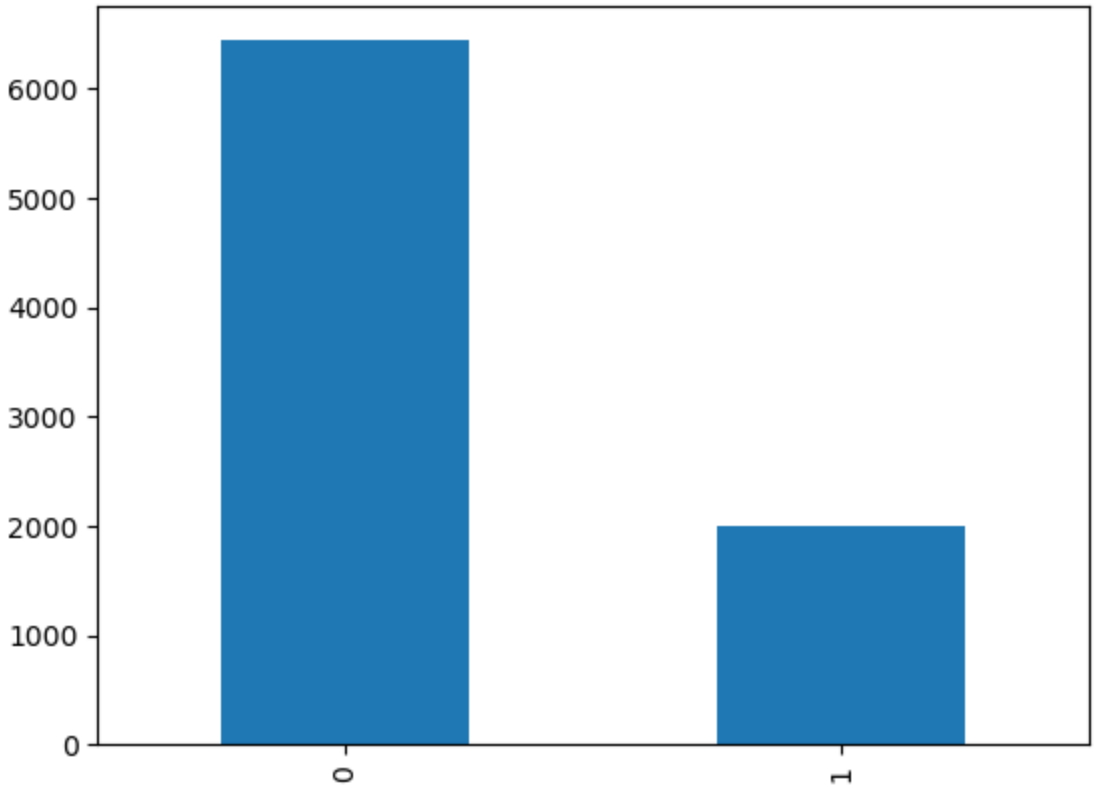
Out[19]:
```
MinTemp           75
MaxTemp           60
Rainfall         240
Evaporation     3512
Sunshine        3994
WindGustSpeed    991
WindSpeed9am      76
WindSpeed3pm     107
Humidity9am       59
Humidity3pm      102
Pressure9am     1309
Pressure3pm     1312
Cloud9am        2421
Cloud3pm        2455
Temp9am           56
Temp3pm           96
year               0
month              0
day                0
dtype: int64
```

In [21]:
```
features_with_outliers = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'WindGustSp
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
```

In [22]:
```
numerical_features_with_null = [feature for feature in numerical_features if df[featur
for feature in numerical_features_with_null:
    mean_value = df[feature].mean()
    df[feature].fillna(mean_value,inplace=True)
```

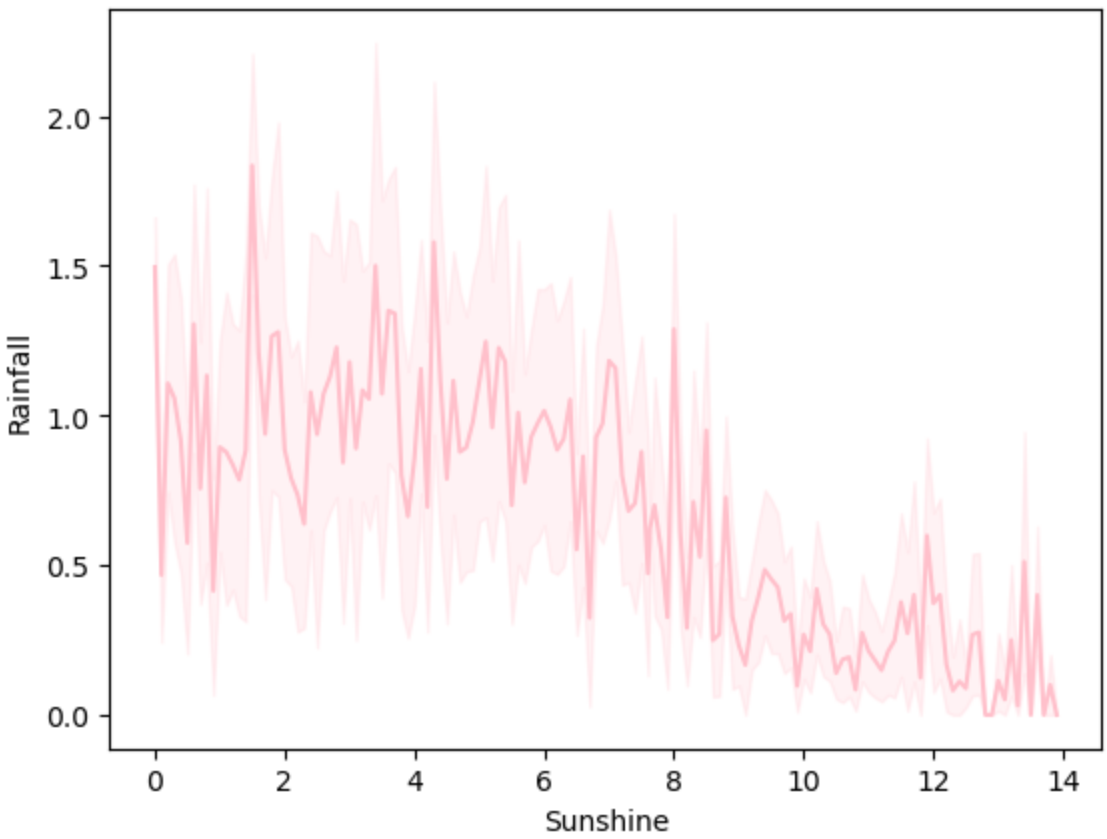
In [28]:
```
df['RainTomorrow'].value_counts().plot(kind='bar')
```

Out[28]:
```
<Axes: >
```



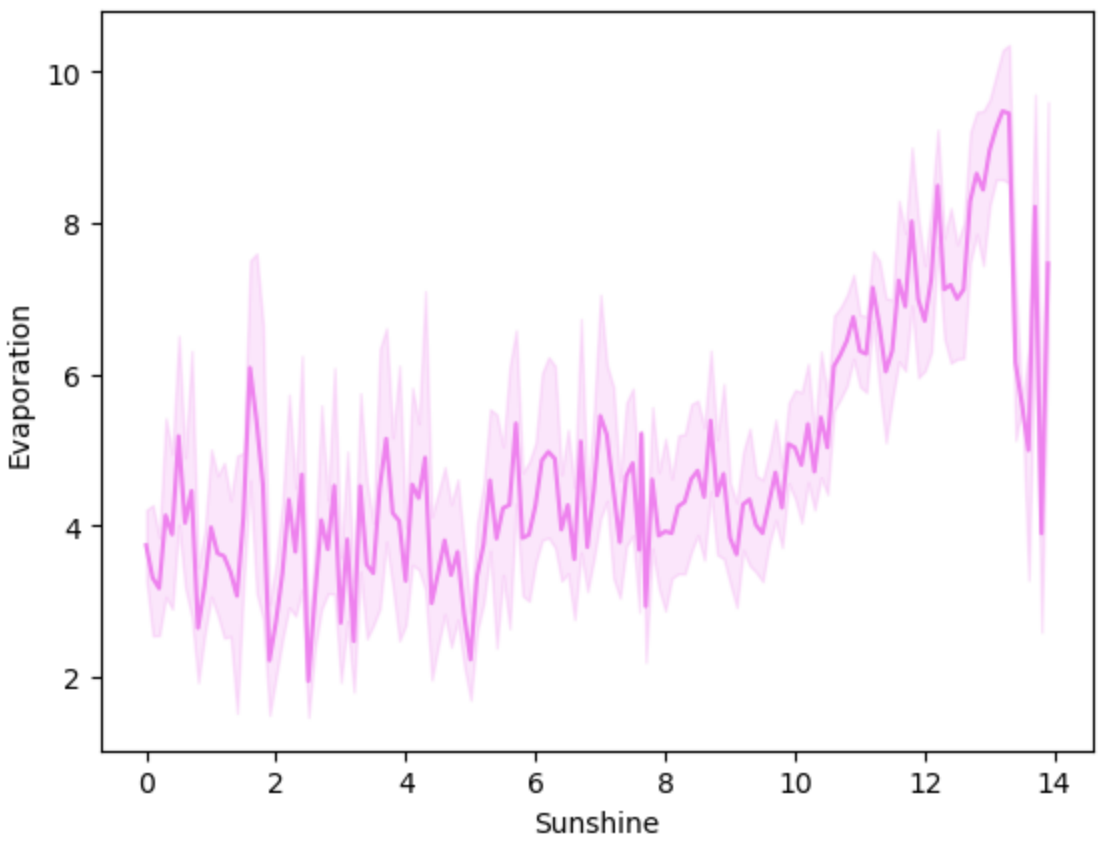
In [30]:
```
sb.lineplot(data=df,x='Sunshine',y='Rainfall',color='pink')
```

Out[30]:
```
<Axes: xlabel='Sunshine', ylabel='Rainfall'>
```



In [31]:
```
sb.lineplot(data=df,x='Sunshine',y='Evaporation',color='violet')
```

Out[31]: <Axes: xlabel='Sunshine', ylabel='Evaporation'>



In [ ]:

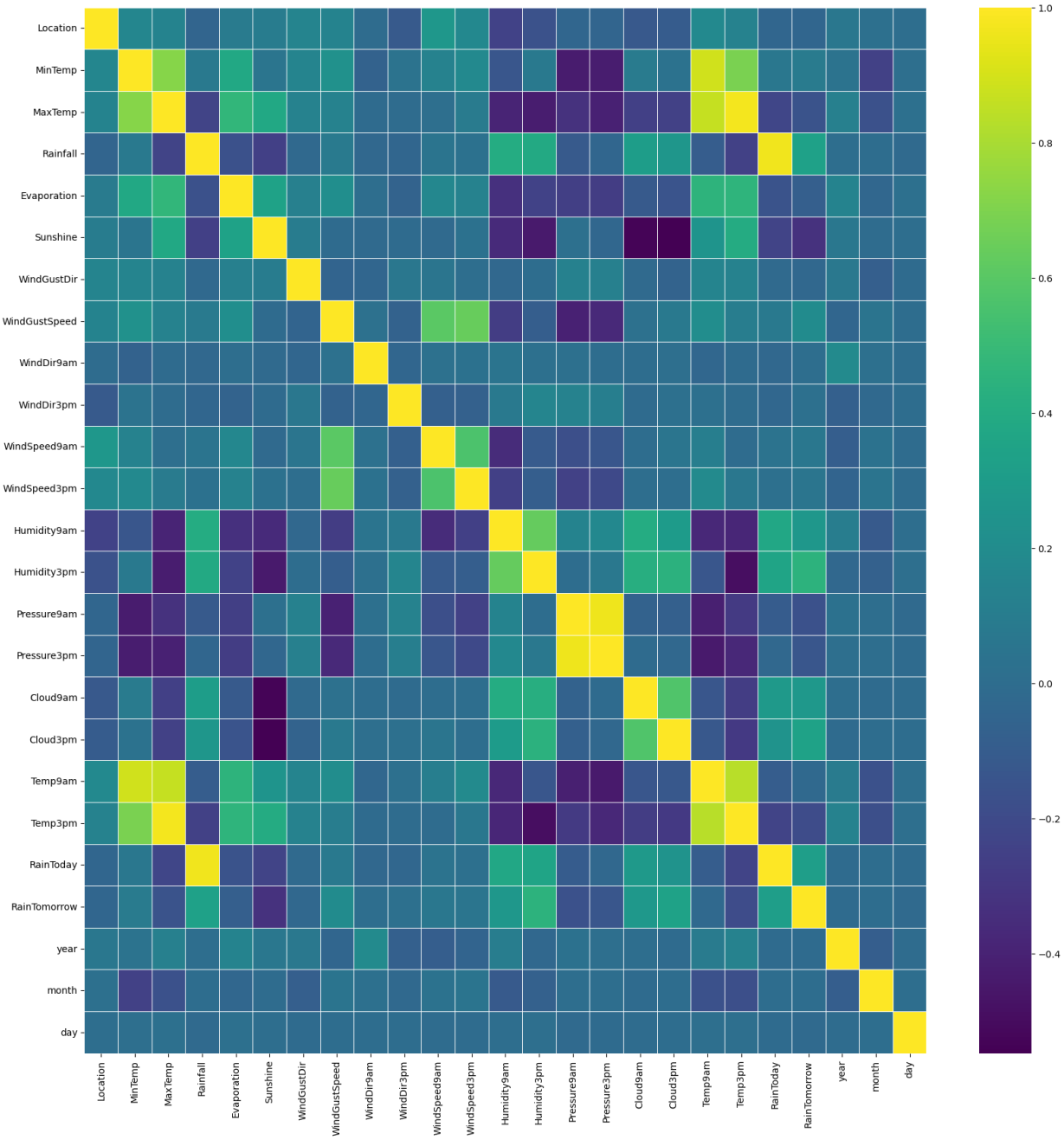
In [ ]:

In [27]:
```
def encode_data(feature_name):


    mapping_dict = {}
    unique_values = list(df[feature_name].unique())
    for idx in range(len(unique_values)):
        mapping_dict[unique_values[idx]] = idx
    return mapping_dict
df['RainToday'].replace({'No':0, 'Yes': 1}, inplace = True)
df['RainTomorrow'].replace({'No':0, 'Yes': 1}, inplace = True)
df['WindGustDir'].replace(encode_data('WindGustDir'),inplace = True)
df['WindDir9am'].replace(encode_data('WindDir9am'),inplace = True)
df['WindDir3pm'].replace(encode_data('WindDir3pm'),inplace = True)
df['Location'].replace(encode_data('Location'), inplace = True)
```

```
plt.figure(figsize=(20,20))
sb.heatmap(df.corr(), linewidths=0.5, annot=False, fmt=".2f", cmap = 'viridis')
```

Out[27]: <Axes: >



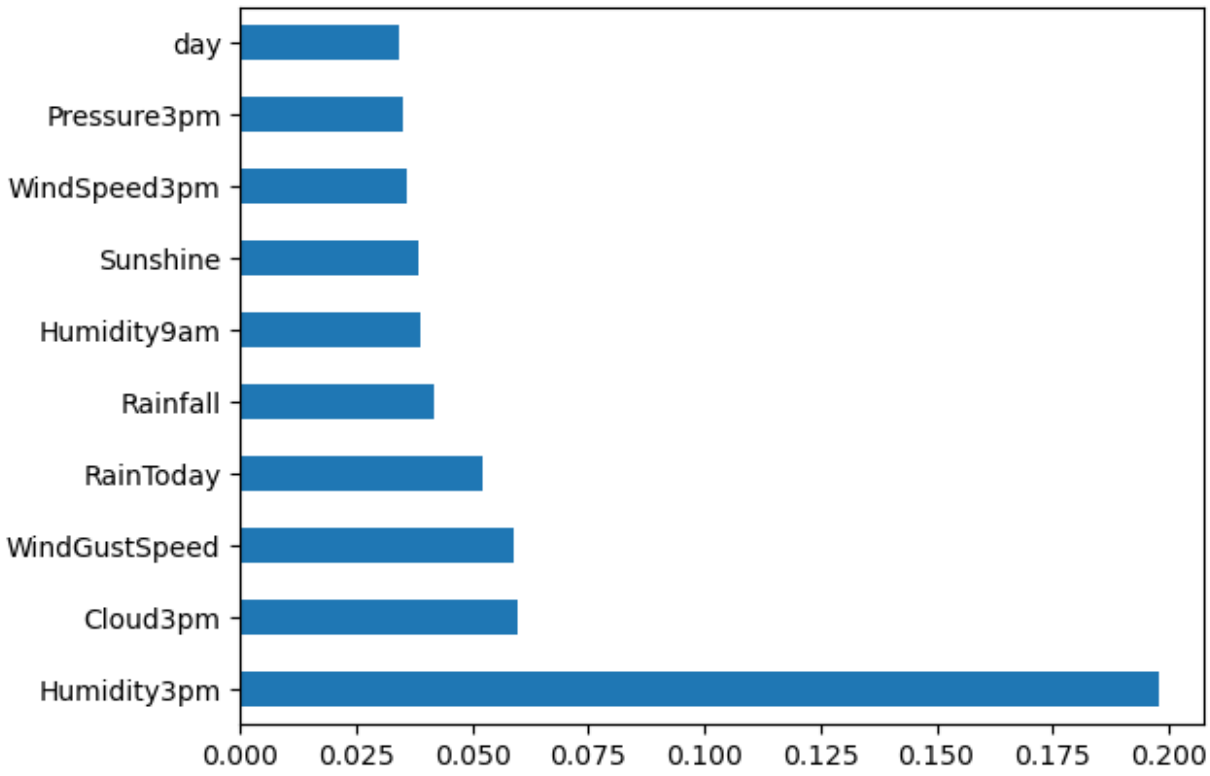
In [32]:
```
X = df.drop(['RainTomorrow'],axis=1)
y = df['RainTomorrow']
```

In [33]:
```
from sklearn.ensemble import ExtraTreesRegressor
etr_model = ExtraTreesRegressor()
etr_model.fit(X,y)
etr_model.feature_importances_
```

Out[33]:
```
array([0.02598707, 0.0273152 , 0.02734835, 0.04172317, 0.02412205,
       0.03825583, 0.02819447, 0.0587736 , 0.03227626, 0.03384829,
       0.03416119, 0.035821  , 0.0387963 , 0.1978236 , 0.03049501,
       0.0348755 , 0.02879411, 0.05986876, 0.02569426, 0.02898582,
       0.05224871, 0.03034129, 0.02994465, 0.0343055 ])
```

In [34]:
```
feature_imp = pd.Series(etr_model.feature_importances_,index=X.columns)
feature_imp.nlargest(10).plot(kind='barh')
```

Out[34]:
```
<Axes: >
```



In [35]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state
```

In [36]:
```
print("Length of Training Data: {}".format(len(X_train)))
print("Length of Testing Data: {}".format(len(X_test)))
```

```
Length of Training Data: 6740
Length of Testing Data: 1685
```

In [37]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
classifier_logreg = LogisticRegression(solver='liblinear', random_state=0)
classifier_logreg.fit(X_train, y_train)
```

Out[38]:
```
LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

In [39]:
```
y_pred = classifier_logreg.predict(X_test)
y_pred
```

Out[39]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

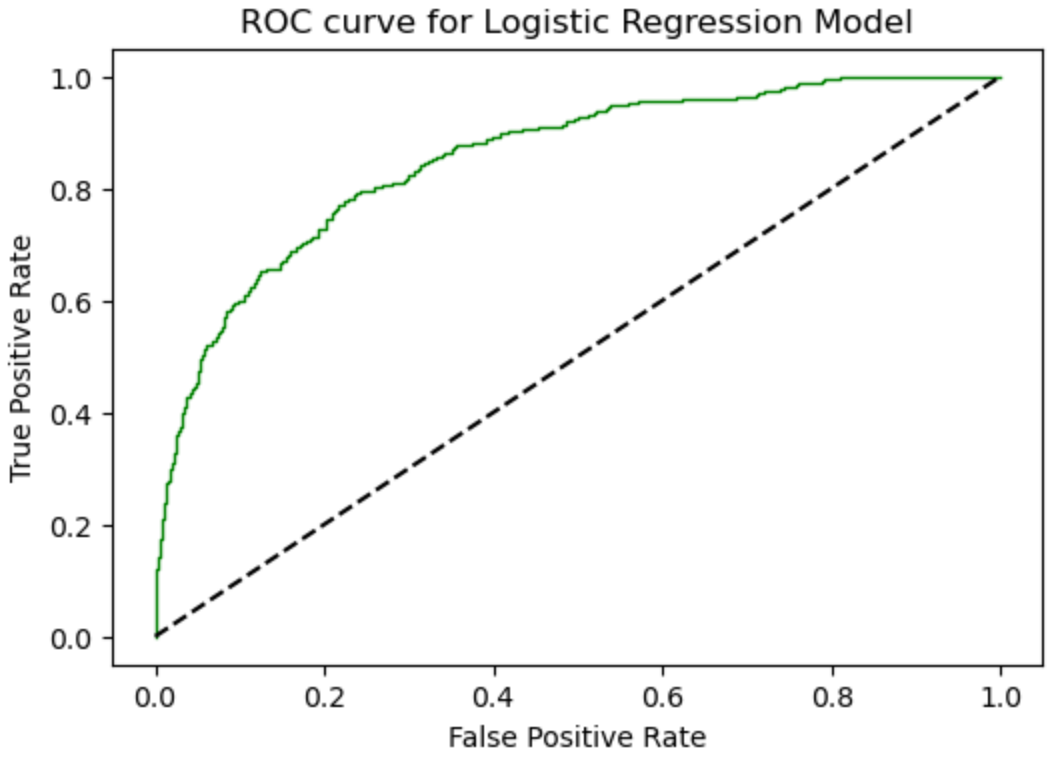
In [40]:
```
from sklearn.metrics import accuracy_score
print("Accuracy Score: {}".format(accuracy_score(y_test,y_pred)))
```

```
Accuracy Score: 0.8367952522255193
```

In [41]:
```
print("Train Data Score: {}".format(classifier_logreg.score(X_train, y_train)))
print("Test Data Score: {}".format(classifier_logreg.score(X_test, y_test)))
```

```
Train Data Score: 0.8302670623145401
Test Data Score: 0.8367952522255193
```

In [43]:
```
y_pred_logreg_proba = classifier_logreg.predict_proba(X_test)
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_logreg_proba[:,1])
plt.figure(figsize=(6,4))
plt.plot(fpr,tpr,'-g',linewidth=1)
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for Logistic Regression Model')
plt.xlabel("False Positive Rate")
plt.ylabel('True Positive Rate')
plt.show()
```



In [44]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(classifier_logreg, X_train, y_train, cv = 5, scoring='accurac
print('Cross-validation scores:{}'.format(scores))
print('Average cross-validation score: {}'.format(scores.mean()))
```

```
Cross-validation scores:[0.82937685 0.83456973 0.83308605 0.83679525 0.81973294]
Average cross-validation score: 0.8307121661721067
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: