IMPPORTING THE LIBRARIES

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

UPLOADING WEATHER CSV

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

CHECKING FIRST AND LAST FIVE ROWS, SHAPE, AND INFO OF THE DATA

In [3]:
```
df.head()
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Win
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	

5 rows × 23 columns

In [4]:
```
df.tail()
```

Out[4]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	NaN

5 rows × 23 columns

In [5]:
```
df.shape
```

Out[5]:
```
(145460, 23)
```

In [6]:
```
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
```

DATA CLEANING STARTING STEPS

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
0
```

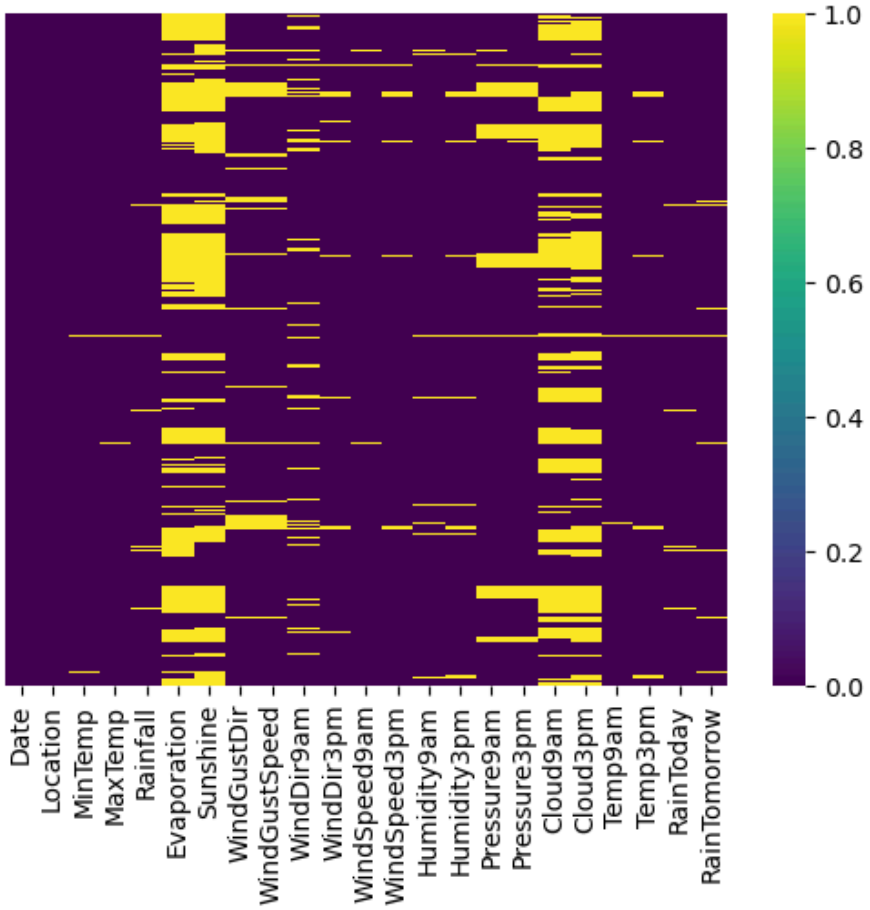
STEP 2 .CHECKING FOR MISSING VALUES

In [8]:
```
df.isna().sum()
```

Out[8]:
```
Date                0
Location            0
MinTemp          1485
MaxTemp          1261
Rainfall         3261
Evaporation     62790
Sunshine        69835
WindGustDir     10326
WindGustSpeed   10263
WindDir9am      10566
WindDir3pm       4228
WindSpeed9am     1767
WindSpeed3pm     3062
Humidity9am      2654
Humidity3pm      4507
Pressure9am     15065
Pressure3pm     15028
Cloud9am        55888
Cloud3pm        59358
Temp9am          1767
Temp3pm          3609
RainToday        3261
RainTomorrow     3267
dtype: int64
```

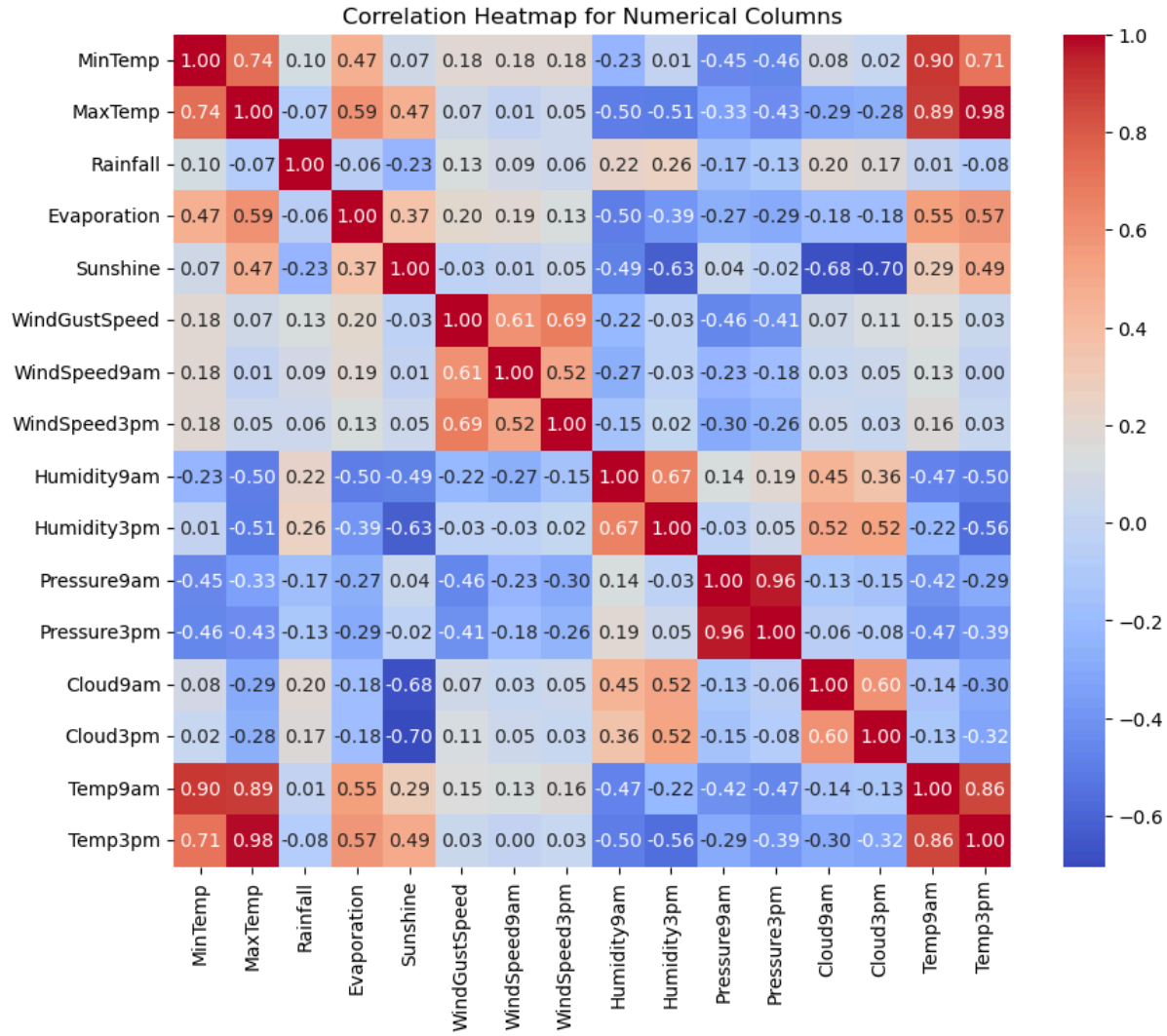
CHECKING MISSING VALUES IN GRAPH

In [9]:
```
sns.heatmap(df.isnull(), cmap='viridis', yticklabels= False)
plt.show()
```



In [10]:
```
# CHECK CORRELATION OF THE COLUMNS
```

In [11]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your dataframe with float and object columns

# Select only numerical columns
numerical_df = df.select_dtypes(include=['float', 'int'])

# Calculate correlation matrix
correlation_matrix = numerical_df.corr()

# Create heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap for Numerical Columns')
plt.show()
```



In [12]:
```
#subshine ,evaporation,cloud3pm,cloud9am are having weak correlation with rainfall,will
```

In [13]:
```
df=df.drop('Evaporation',axis=1)
```

In [14]:
```
df=df.drop('Sunshine',axis=1)
```

In [15]:
```
df= df.drop('Cloud9am',axis=1)
```

In [16]:
```
df= df.drop('Cloud3pm',axis=1)
```

In [17]:
```
# again check if columns has deleted from data set
```

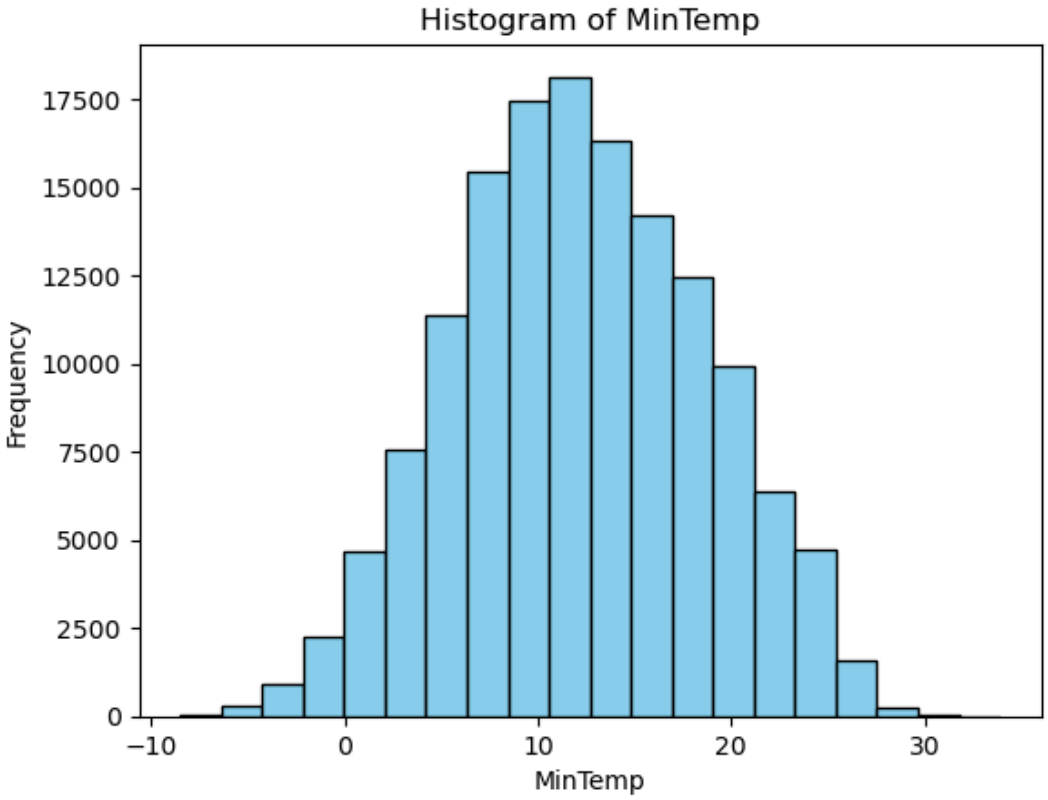
In [18]:
```
df.isna().sum()
```

Out[18]:
```
Date                0
Location            0
MinTemp          1485
MaxTemp          1261
Rainfall         3261
WindGustDir     10326
WindGustSpeed   10263
WindDir9am      10566
WindDir3pm       4228
WindSpeed9am     1767
WindSpeed3pm     3062
Humidity9am      2654
Humidity3pm      4507
Pressure9am     15065
Pressure3pm     15028
Temp9am          1767
Temp3pm          3609
RainToday        3261
RainTomorrow     3267
dtype: int64
```

In [19]:
```
#now decided to fill other values numerical first
```

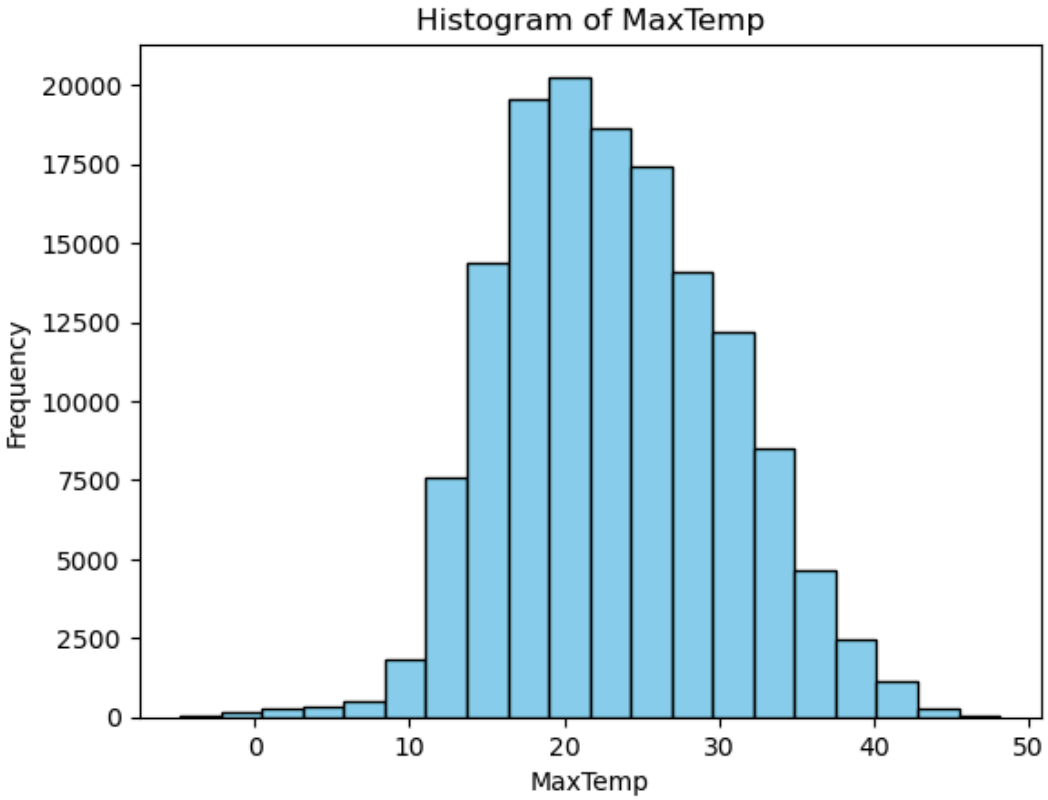
In [20]:
```
#for this first we see their distributions of data
```

In [21]:
```
plt.hist(df['MinTemp'], bins=20, color='skyblue', edgecolor='black')


plt.xlabel('MinTemp')
plt.ylabel('Frequency')
plt.title('Histogram of MinTemp')


plt.show()
```

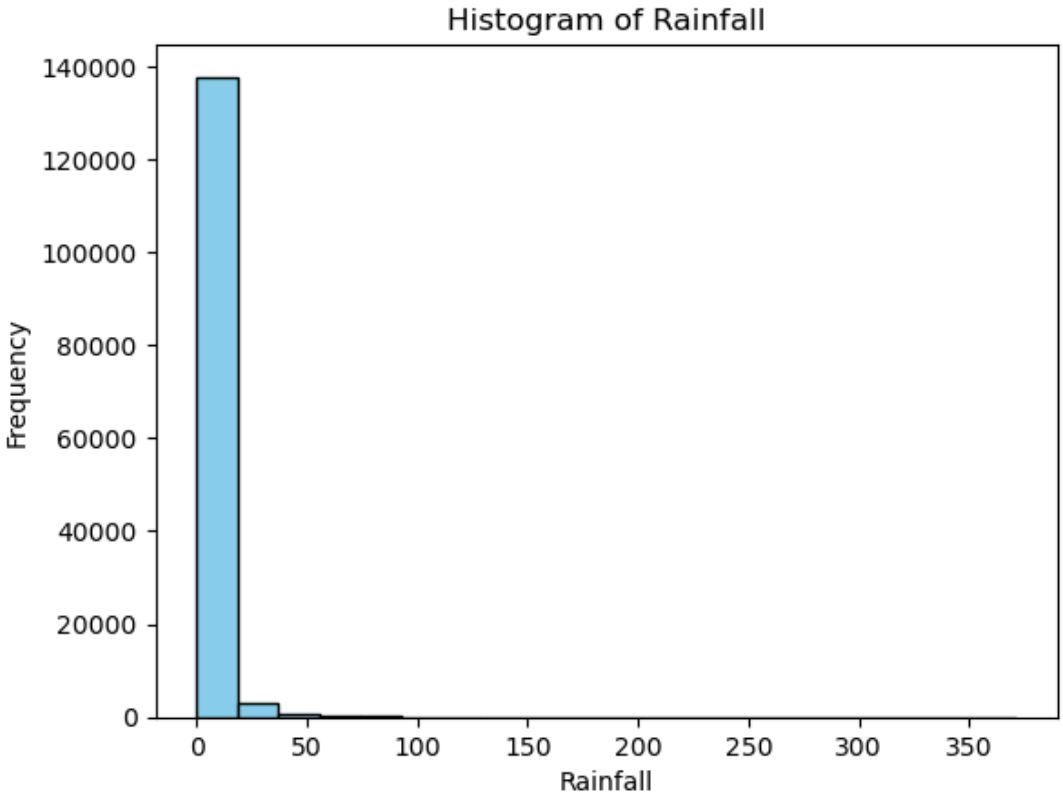


In [22]:

```
plt.hist(df['MaxTemp'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('MaxTemp')
plt.ylabel('Frequency')
plt.title('Histogram of MaxTemp')
plt.show()
```

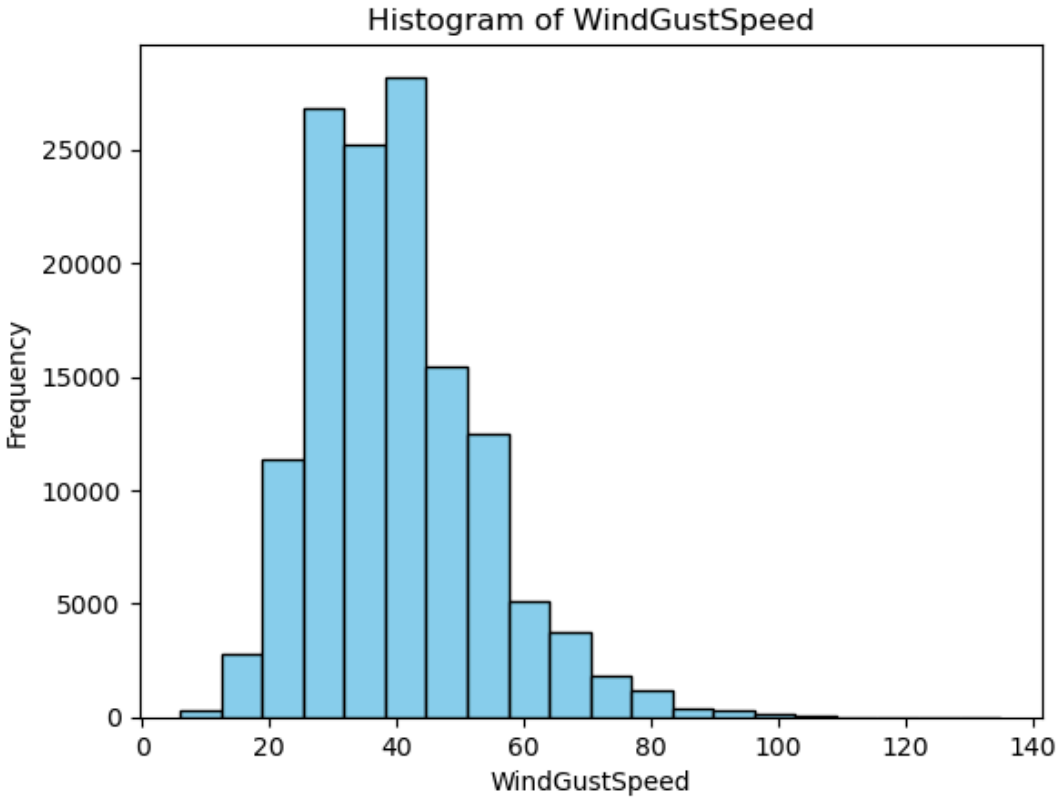


In [23]:
```
plt.hist(df['Rainfall'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Rainfall')
plt.ylabel('Frequency')
plt.title('Histogram of Rainfall')
plt.show()
```



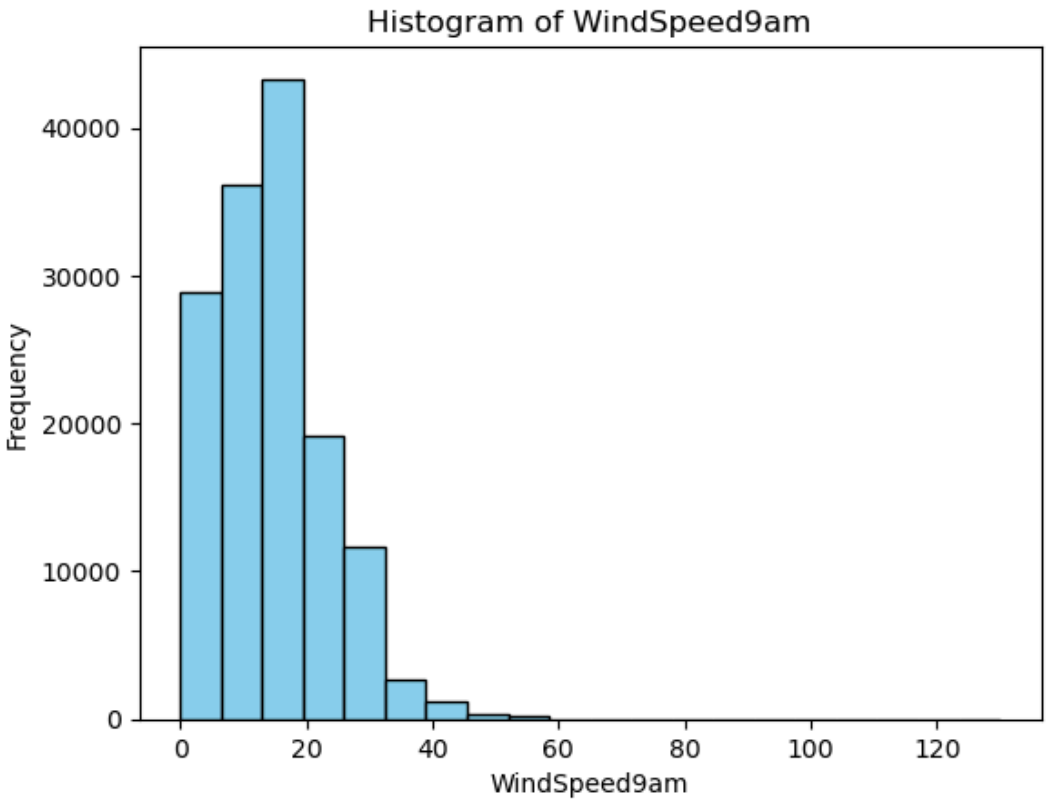
In [24]:

```
plt.hist(df['WindGustSpeed'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('WindGustSpeed')
plt.ylabel('Frequency')
plt.title('Histogram of WindGustSpeed')
plt.show()
```

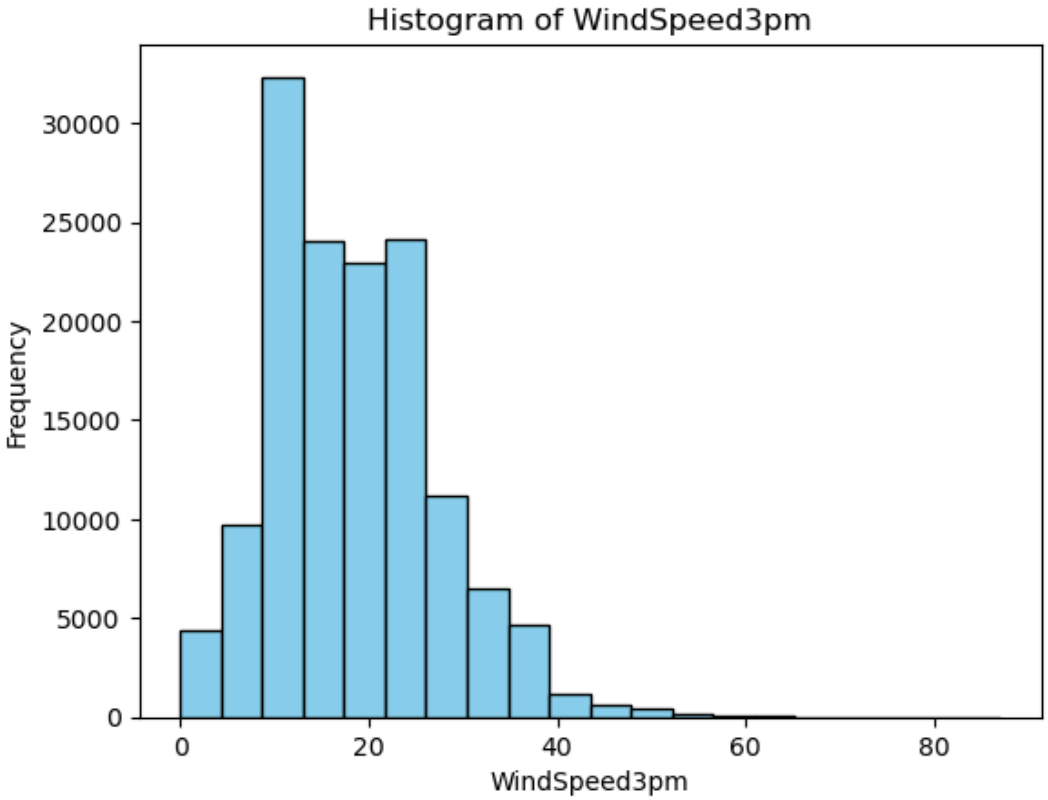


In [25]:

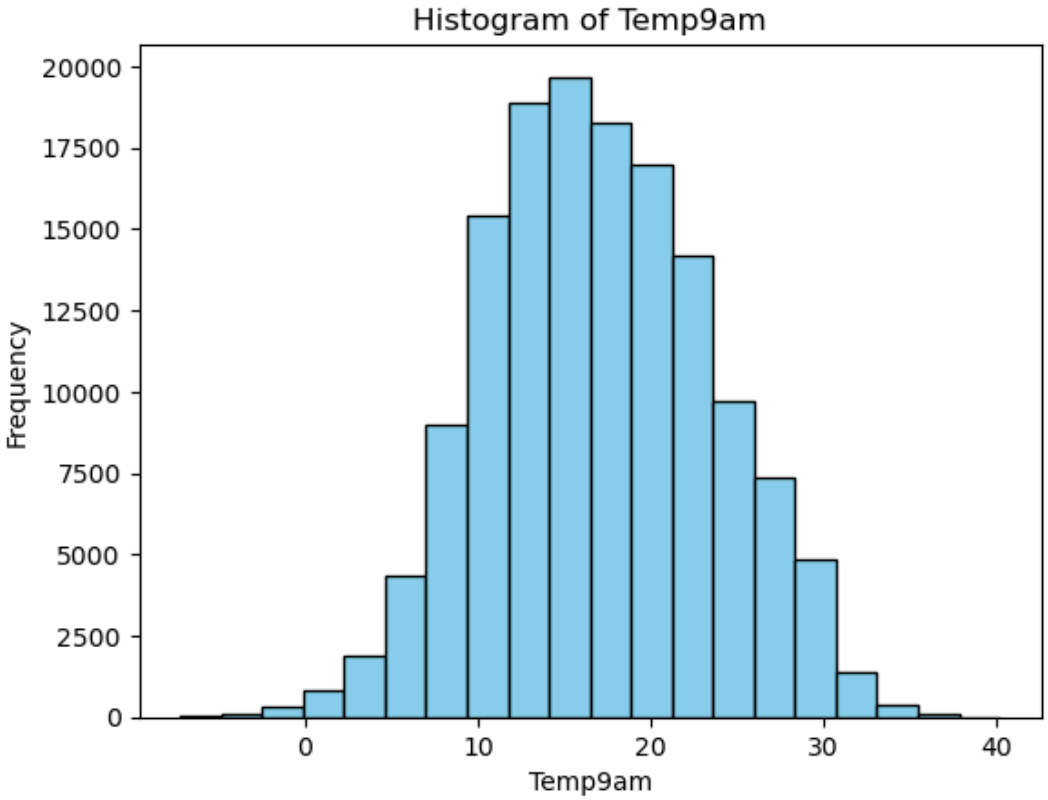
```
plt.hist(df['WindSpeed9am'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('WindSpeed9am')
plt.ylabel('Frequency')
plt.title('Histogram of WindSpeed9am')
plt.show()
```



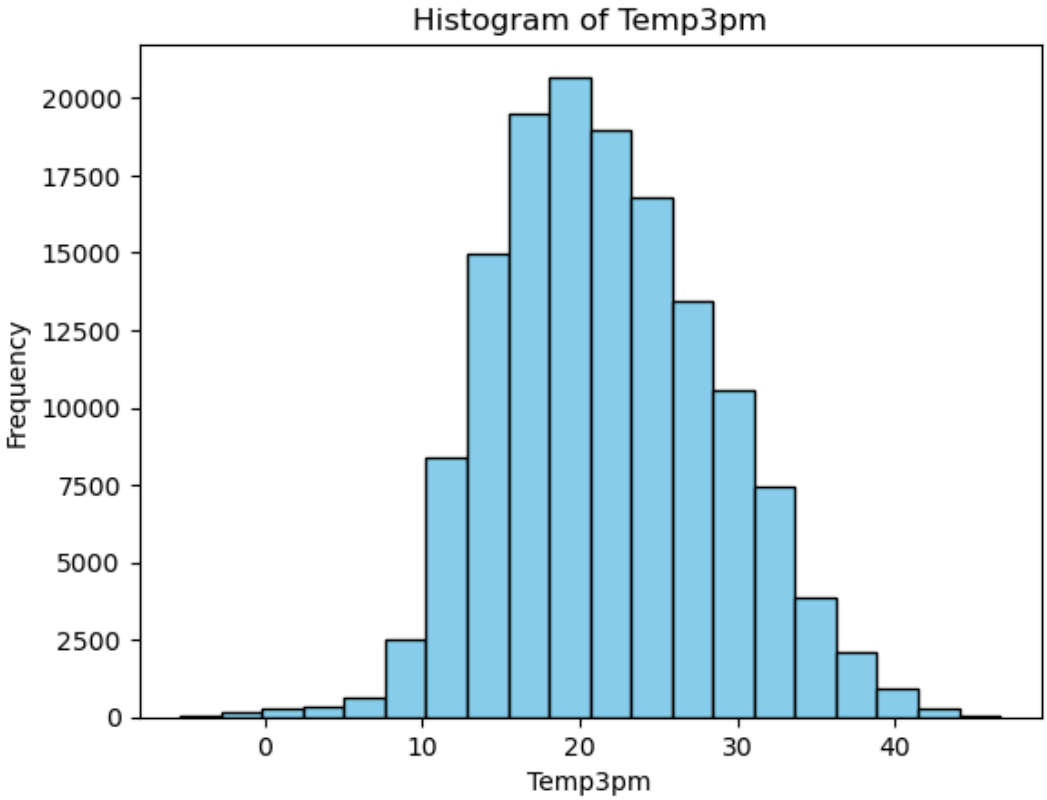
In [26]:
```
plt.hist(df['WindSpeed3pm'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('WindSpeed3pm')
plt.ylabel('Frequency')
plt.title('Histogram of WindSpeed3pm')
plt.show()
```



In [27]:
```
plt.hist(df['Temp9am'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Temp9am')
plt.ylabel('Frequency')
plt.title('Histogram of Temp9am')
plt.show()
```



In [28]:
```
plt.hist(df['Temp3pm'], bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Temp3pm')
plt.ylabel('Frequency')
plt.title('Histogram of Temp3pm')
plt.show()
```



In [29]:
```
# we have see other then rainfall in all we have to replace median values
```

In [30]:
```
#now will fill the missing values in all numerical columns
```

In [31]:
```
df['Temp3pm'].fillna(df['Temp3pm'].median(), inplace=True)
```

In [32]:
```
df['Temp9am'].fillna(df['Temp9am'].median(), inplace=True)
```

In [33]:
```
df['WindSpeed3pm'].fillna(df['WindSpeed3pm'].median(), inplace=True)
```

In [34]:
```
df['WindSpeed9am'].fillna(df['WindSpeed9am'].median(), inplace=True)
```

In [ ]:
```

```

In [35]:
```
df['MinTemp'].fillna(df['MinTemp'].median(), inplace=True)
```

In [36]:
```
df['MaxTemp'].fillna(df['MaxTemp'].median(), inplace=True)
```

In [37]:
```
df['Humidity9am'].fillna(df['Humidity9am'].median(), inplace=True)
```

In [38]:
```
df['Humidity3pm'].fillna(df['Humidity3pm'].median(), inplace=True)
```

In [39]:
```
df['Pressure9am'].fillna(df['Pressure9am'].median(), inplace=True)
```

In [40]:
```
df['Pressure3pm'].fillna(df['Pressure3pm'].median(), inplace=True)
```

In [41]:
```
df['Temp3pm'].fillna(df['Temp3pm'].median(), inplace=True)
```

In [42]:
```
df['Temp9am'].fillna(df['Temp9am'].median(), inplace=True)
```

In [43]:
```
df['Rainfall']=df['Rainfall'].fillna(0)
```

In [44]:
```
df['WindGustSpeed'].fillna(df['WindGustSpeed'].median(), inplace=True)
```

In [45]:
```
# again check if missing values are filled in data numerical columns

```

In [46]:
```
df.isna().sum()
```

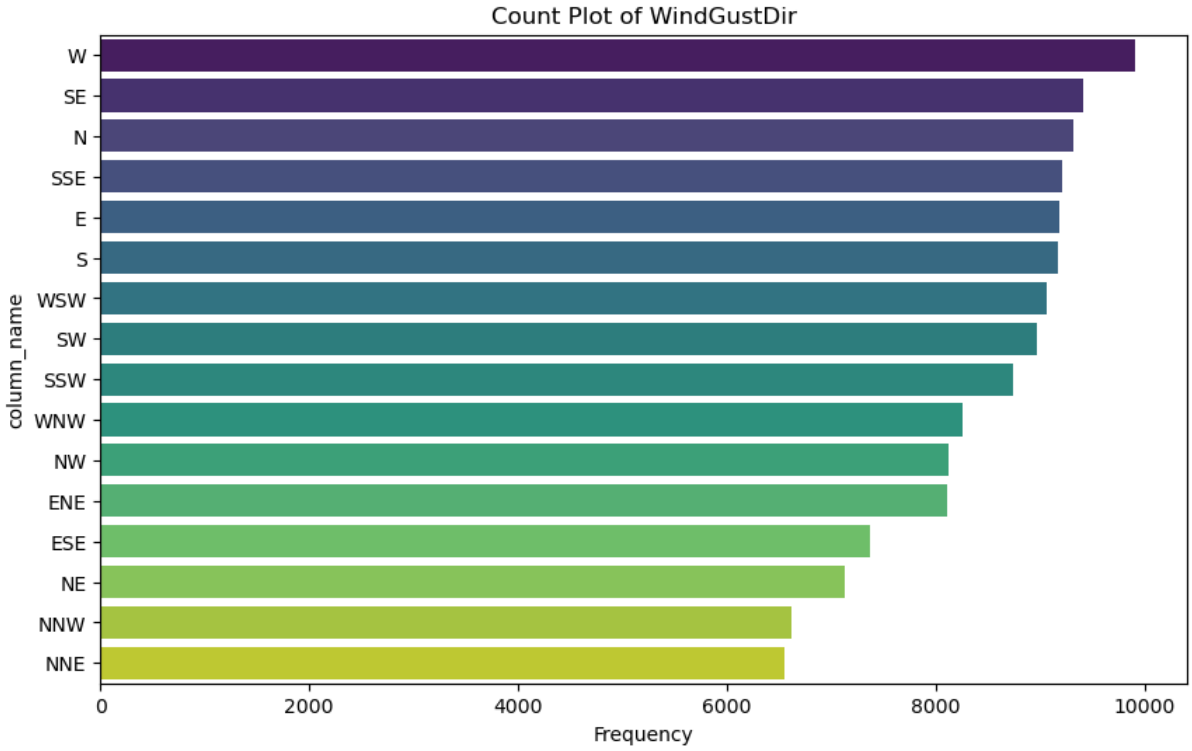
Out[46]:
```
Date                0
Location            0
MinTemp             0
MaxTemp             0
Rainfall            0
WindGustDir     10326
WindGustSpeed       0
WindDir9am      10566
WindDir3pm       4228
WindSpeed9am        0
WindSpeed3pm        0
Humidity9am         0
Humidity3pm         0
Pressure9am         0
Pressure3pm         0
Temp9am             0
Temp3pm             0
RainToday        3261
RainTomorrow     3267
dtype: int64
```

In [47]:
```
df.info()
```

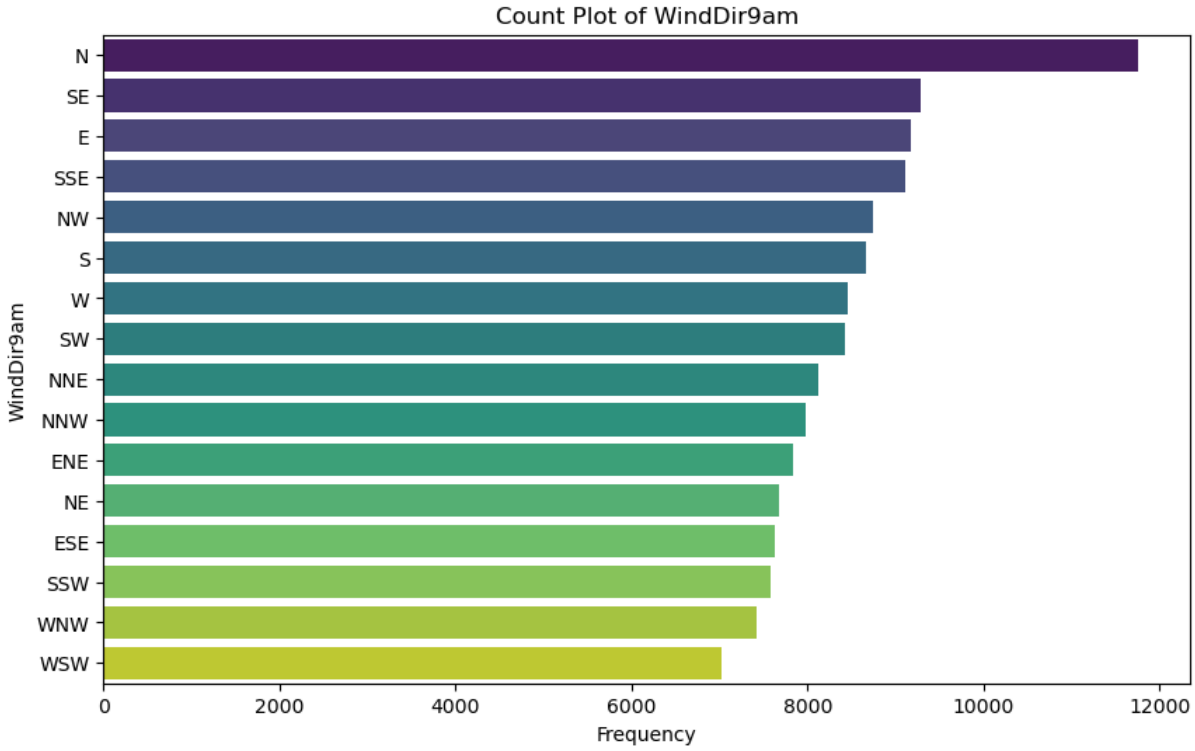
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 19 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           145460 non-null  object 
 1   Location       145460 non-null  object 
 2   MinTemp        145460 non-null  float64
 3   MaxTemp        145460 non-null  float64
 4   Rainfall       145460 non-null  float64
 5   WindGustDir    135134 non-null  object 
 6   WindGustSpeed  145460 non-null  float64
 7   WindDir9am     134894 non-null  object 
 8   WindDir3pm     141232 non-null  object 
 9   WindSpeed9am   145460 non-null  float64
 10  WindSpeed3pm   145460 non-null  float64
 11  Humidity9am    145460 non-null  float64
 12  Humidity3pm    145460 non-null  float64
 13  Pressure9am    145460 non-null  float64
 14  Pressure3pm    145460 non-null  float64
 15  Temp9am        145460 non-null  float64
 16  Temp3pm        145460 non-null  float64
 17  RainToday      142199 non-null  object 
 18  RainTomorrow   142193 non-null  object 
dtypes: float64(12), object(7)
memory usage: 21.1+ MB
```

In [48]:
```
# now we have to deal object type columns,that noramlly filled with mode,but firstle we
```

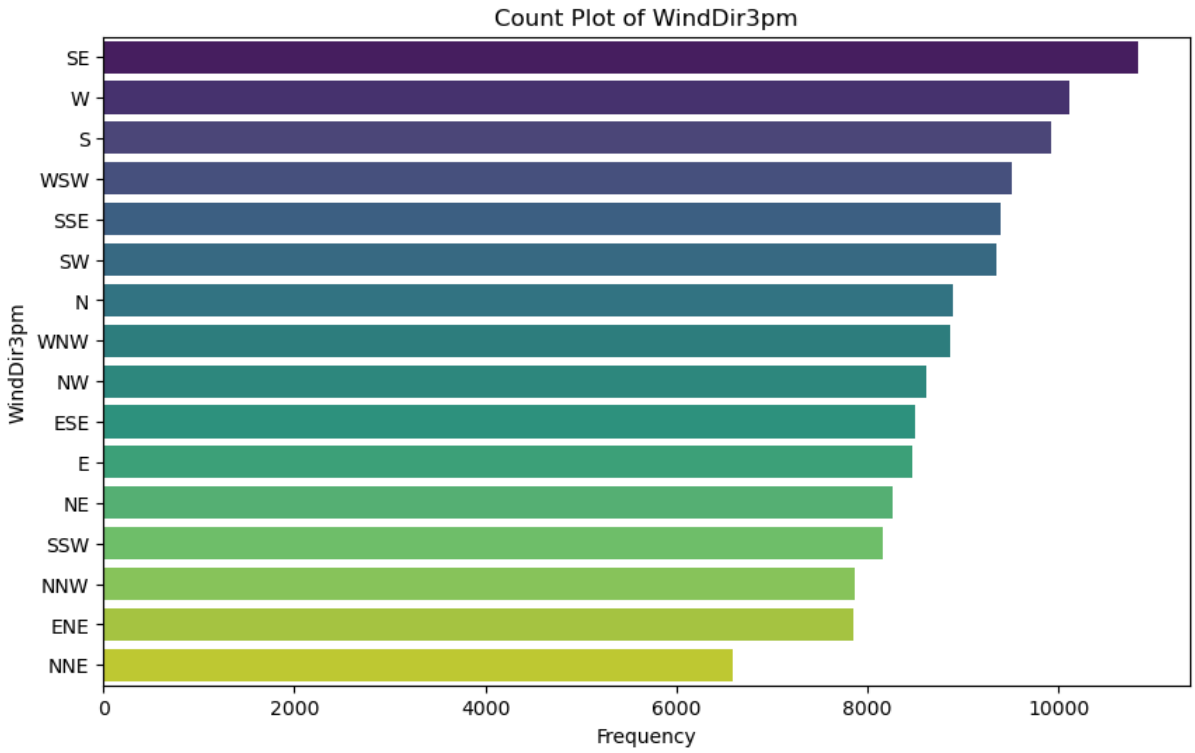
In [51]:
```
import seaborn as sns

# Assuming df is your DataFrame and 'windgustdir' is the object-type column you want to
column_name = 'WindGustDir'

# Plot count plot
plt.figure(figsize=(10, 6))
sns.countplot(y=df[column_name], order=df[column_name].value_counts().index, palette='v

# Add labels and title
plt.xlabel('Frequency')
plt.ylabel('column_name')
plt.title('Count Plot of WindGustDir')

# Show plot
plt.show()
```



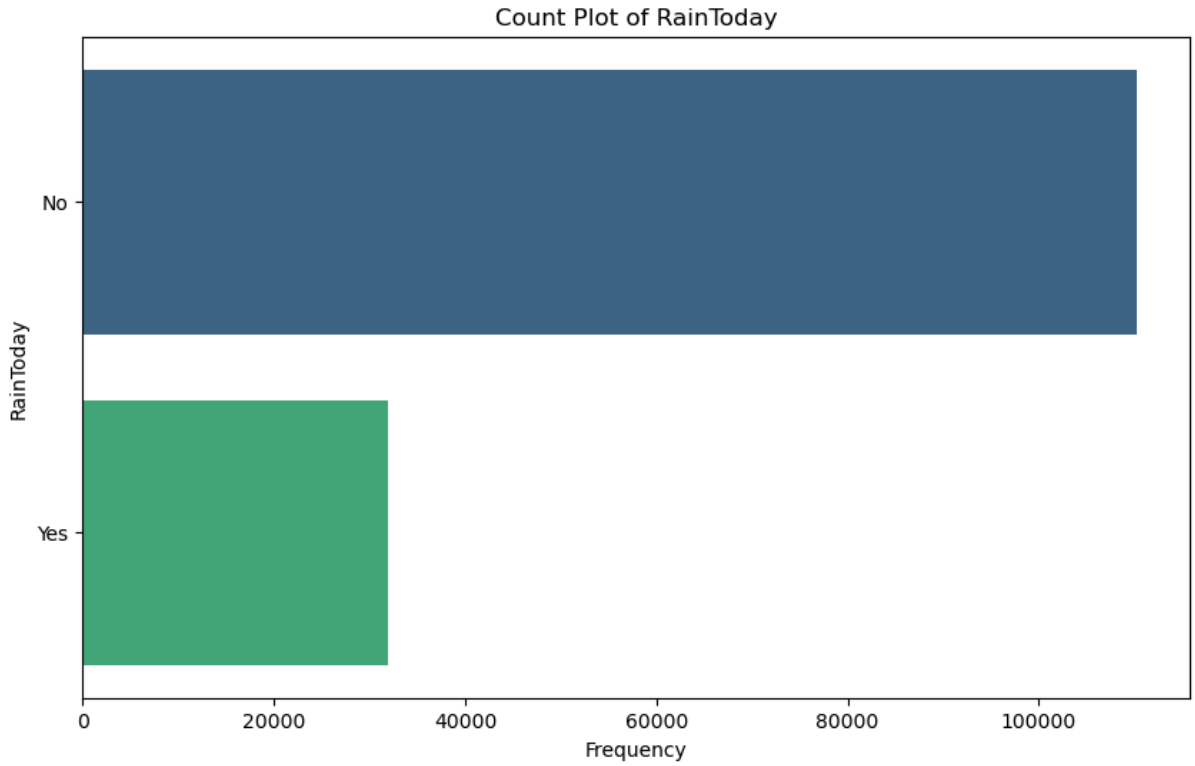
In [52]:
```
import seaborn as sns

# Assuming df is your DataFrame and 'windgustdir' is the object-type column you want to
column_name = 'WindDir9am'

# Plot count plot
plt.figure(figsize=(10, 6))
sns.countplot(y=df[column_name], order=df[column_name].value_counts().index, palette='v

# Add labels and title
plt.xlabel('Frequency')
plt.ylabel(column_name)
plt.title('Count Plot of WindDir9am')

# Show plot
plt.show()
```



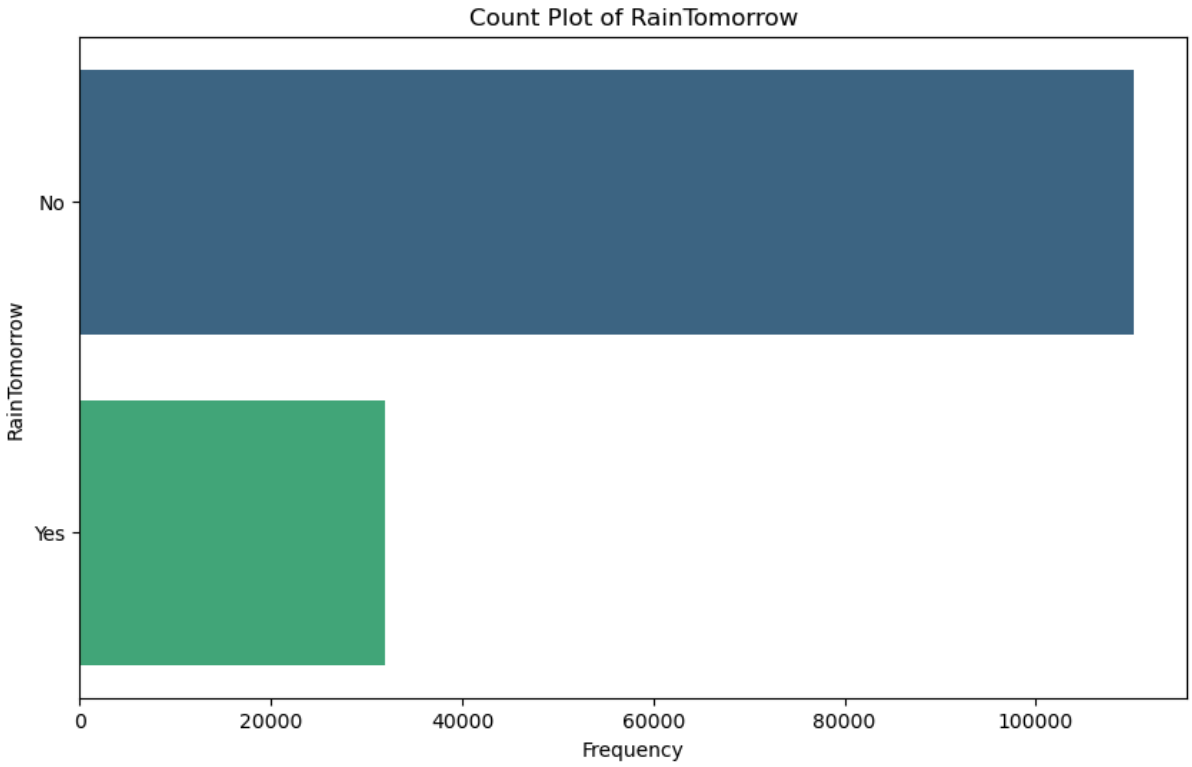
In [53]:
```
import seaborn as sns

# Assuming df is your DataFrame and 'windgustdir' is the object-type column you want to
column_name = 'WindDir3pm'

# Plot count plot
plt.figure(figsize=(10, 6))
sns.countplot(y=df[column_name], order=df[column_name].value_counts().index, palette='v

# Add labels and title
plt.xlabel('Frequency')
plt.ylabel(column_name)
plt.title('Count Plot of WindDir3pm')

# Show plot
plt.show()
```



In [54]:
```
# Assuming df is your DataFrame and 'windgustdir' is the object-type column you want to
column_name = 'RainToday'

# Plot count plot
plt.figure(figsize=(10, 6))
sns.countplot(y=df[column_name], order=df[column_name].value_counts().index, palette='v

# Add labels and title
plt.xlabel('Frequency')
plt.ylabel(column_name)
plt.title('Count Plot of RainToday')

# Show plot
plt.show()
```



In [55]:
```
# Assuming df is your DataFrame and 'windgustdir' is the object-type column you want to
column_name = 'RainTomorrow'

# Plot count plot
plt.figure(figsize=(10, 6))
sns.countplot(y=df[column_name], order=df[column_name].value_counts().index, palette='v

# Add labels and title
plt.xlabel('Frequency')
plt.ylabel(column_name)
plt.title('Count Plot of RainTomorrow')

# Show plot
plt.show()
```



In [56]:
```
#now fill values there in no repetititon of mode,there can be more mode used iloc[0] to
```

In [57]:
```
df['RainToday'].fillna(df['RainToday'].mode().iloc[0], inplace=True)
```

In [58]:
```
df['RainTomorrow'].fillna(df['RainTomorrow'].mode().iloc[0], inplace=True)
```

In [59]:
```
df['WindDir3pm'].fillna(df['WindDir3pm'].mode().iloc[0], inplace=True)
```

In [60]:
```
df['WindDir9am'].fillna(df['WindDir9am'].mode().iloc[0], inplace=True)
```

In [61]:
```
df['WindGustDir'].fillna(df['WindGustDir'].mode().iloc[0], inplace=True)
```

In [62]:
```
# again check for filling missing values in the data
```

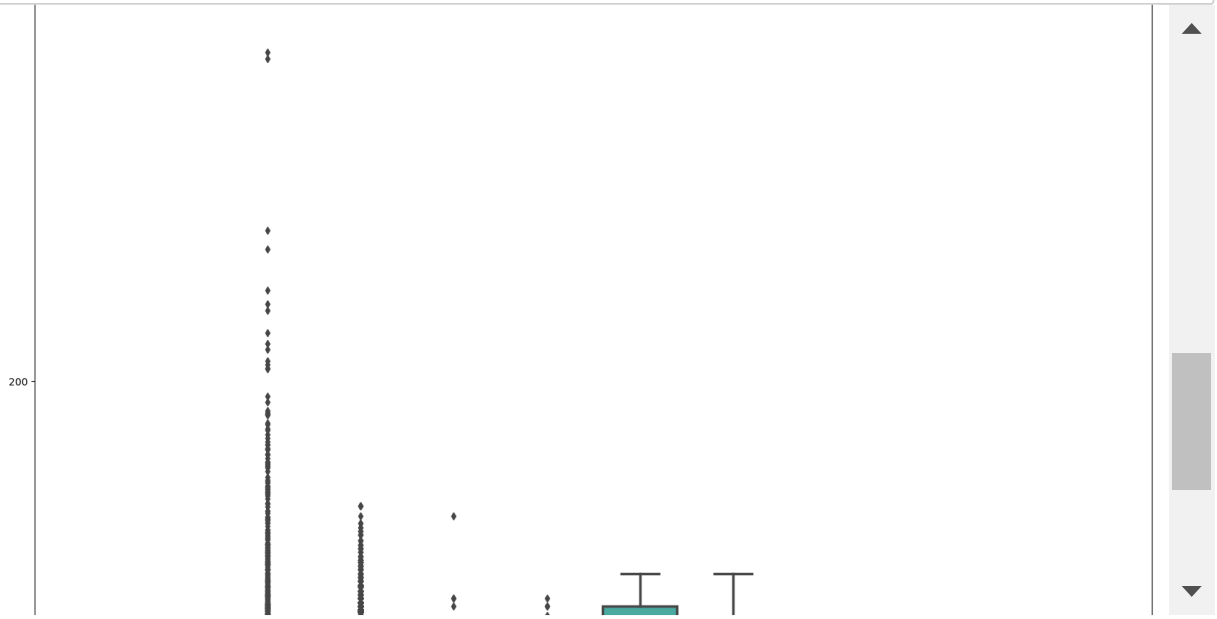
In [63]:
```
df.isna().sum()
```

Out[63]:
```
Date             0
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Temp9am          0
Temp3pm          0
RainToday        0
RainTomorrow     0
dtype: int64
```

In [64]:
```
#finally missing values filled in the data
```

In [65]:
```
#now check for outliers
```

In [66]:
```


# Select only numerical columns
numerical_df = df.select_dtypes(include=['float', 'int'])

# Plot boxplots for each numerical column
plt.figure(figsize=(20, 40))     # 30 is representing to increase the size of box plot
sns.boxplot(data=numerical_df, linewidth=2.5)
plt.title('Boxplot of Numerical Columns')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```



In [67]:
```
# working to remove the outliers from all numerical columns
```

In [68]:
```
# Assuming df is your dataframe

# Select only numerical columns
numerical_df = df.select_dtypes(include=['float', 'int'])

# Calculate the quartiles
Q1 = numerical_df.quantile(0.25)
Q3 = numerical_df.quantile(0.75)

# Calculate the IQR
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
df_no_outliers = df[~((numerical_df < lower_bound) | (numerical_df > upper_bound)).any(
```

In [69]:
```
df.describe()
```

Out[69]:

	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Hur
count	145460.000000	145460.000000	145460.000000	145460.000000	145460.000000	145460.000000	1454
mean	12.192053	23.215962	2.307990	39.962189	14.030751	18.669758	
std	6.365780	7.088358	8.389771	13.120931	8.861796	8.716716	
min	-8.500000	-4.800000	0.000000	6.000000	0.000000	0.000000	
25%	7.700000	18.000000	0.000000	31.000000	7.000000	13.000000	
50%	12.000000	22.600000	0.000000	39.000000	13.000000	19.000000	
75%	16.800000	28.200000	0.600000	46.000000	19.000000	24.000000	
max	33.900000	48.100000	371.000000	135.000000	130.000000	87.000000	1

In [70]:
```
df_no_outliers.describe()
```

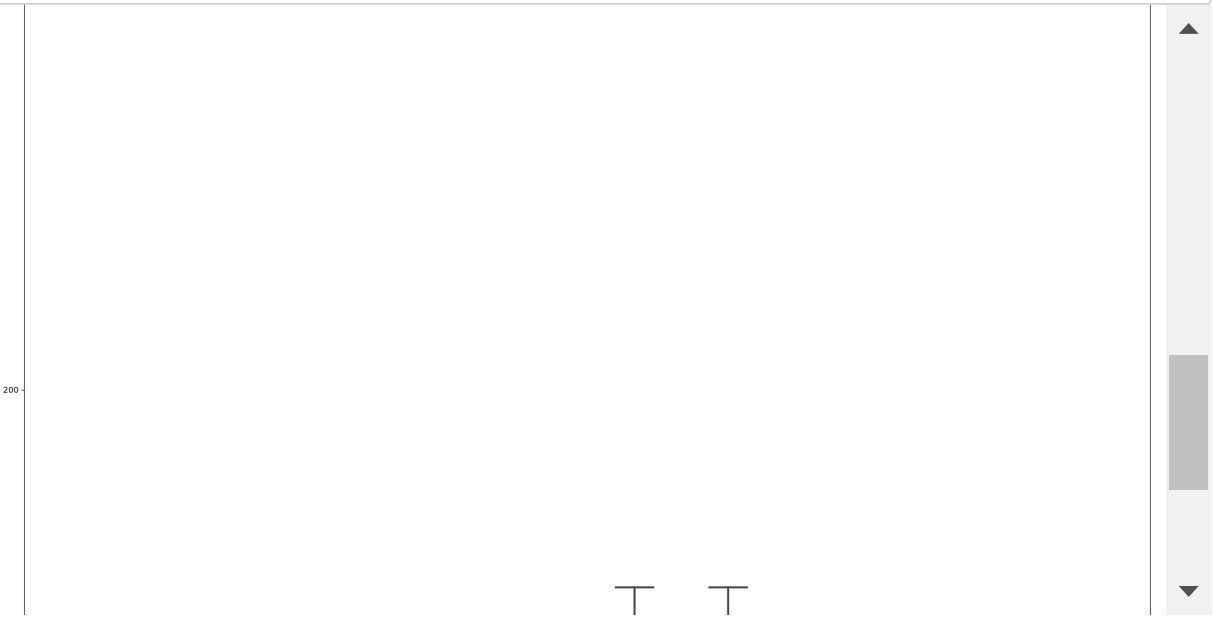
Out[70]:

	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Hur
count	108029.000000	108029.000000	108029.000000	108029.000000	108029.000000	108029.000000	1080
mean	11.963318	23.891871	0.102784	37.525766	12.949736	17.704820	
std	6.347131	6.685311	0.271186	10.442607	7.868914	7.718722	
min	-5.900000	2.800000	0.000000	9.000000	0.000000	0.000000	
25%	7.400000	18.900000	0.000000	30.000000	7.000000	13.000000	
50%	11.900000	23.400000	0.000000	37.000000	13.000000	17.000000	
75%	16.500000	28.700000	0.000000	44.000000	19.000000	22.000000	
max	30.200000	43.500000	1.500000	67.000000	37.000000	39.000000	1

In [71]:
```
#i have seen statistic summary and amzed that outliers are remover
```

In [72]:
```
# Rename the dataframe
updated_df = df_no_outliers
```

In [73]:
```
numerical_df = updated_df.select_dtypes(include=['float', 'int'])

# Plot boxplots for each numerical column
plt.figure(figsize=(20, 40))     # 30 is representing to increase the size of box plot
sns.boxplot(data=numerical_df, linewidth=2.5)
plt.title('Boxplot of Numerical Columns')
plt.xticks(rotation=45)# Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```



In [74]:
```
#now my new data name is df_no_outliers
```

In [75]:
```
#will be using this for stadardization and scalling
```

In [76]:
```
df_no_outliers.head()
```

Out[76]:

	Date	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	\
0	2008-12-01	Albury	13.4	22.9	0.6	W	44.0	W	WNW	
1	2008-12-02	Albury	7.4	25.1	0.0	WNW	44.0	NNW	WSW	
2	2008-12-03	Albury	12.9	25.7	0.0	WSW	46.0	W	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NE	24.0	SE	E	
4	2008-12-05	Albury	17.5	32.3	1.0	W	41.0	ENE	NW	

In [77]:
```
# Rename the dataframe
updated_df = df_no_outliers
```

In [78]:
```
updated_df.shape
```

Out[78]: (108029, 19)

In [79]:
```
#standardised my columns
```

In [80]:
```
from sklearn.preprocessing import StandardScaler

# Assuming updated_df is your dataframe with numerical columns

# Create a StandardScaler object
scaler = StandardScaler()

# Select only numerical columns
numerical_df = updated_df.select_dtypes(include=['float', 'int'])

# Standardize numerical columns
updated_df[numerical_df.columns] = scaler.fit_transform(numerical_df)

# Now numerical columns in updated_df are standardized
```

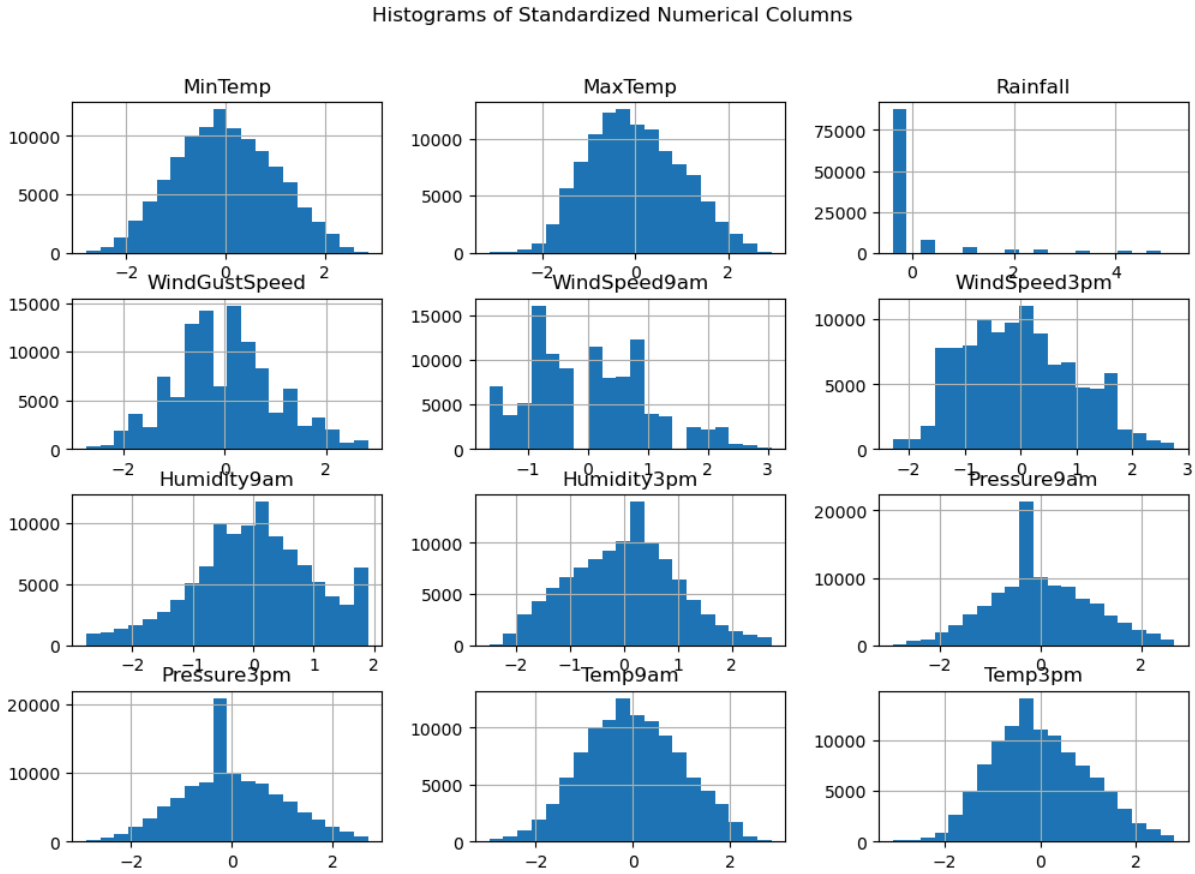
```
C:\Users\hamma\AppData\Local\Temp\ipykernel_29300\4090449474.py:12: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  updated_df[numerical_df.columns] = scaler.fit_transform(numerical_df)
```

In [81]:
```
# Calculate the mean and standard deviation of each numerical column
mean_values = updated_df[numerical_df.columns].mean()
std_values = updated_df[numerical_df.columns].std()

# Print mean and standard deviation values
print("Mean values:\n", mean_values)
print("\nStandard deviation values:\n", std_values)
```

```
Mean values:
 MinTemp         -2.104747e-18
MaxTemp          7.114044e-16
Rainfall        -3.999019e-17
WindGustSpeed    2.304698e-16
WindSpeed9am    -2.315221e-17
WindSpeed3pm    -2.004771e-16
Humidity9am     -2.062652e-16
Humidity3pm     -7.998037e-17
Pressure9am      7.236119e-15
Pressure3pm      3.750659e-15
Temp9am          1.399657e-16
Temp3pm          2.683552e-16
dtype: float64

Standard deviation values:
 MinTemp          1.000005
MaxTemp          1.000005
Rainfall         1.000005
WindGustSpeed    1.000005
WindSpeed9am     1.000005
WindSpeed3pm     1.000005
Humidity9am      1.000005
Humidity3pm      1.000005
Pressure9am      1.000005
Pressure3pm      1.000005
Temp9am          1.000005
Temp3pm          1.000005
dtype: float64
```

In [82]:
```
#checking standardized column
```

In [83]:
```
# Assuming updated_df is your dataframe with standardized numerical columns

# Select only numerical columns
numerical_df = updated_df.select_dtypes(include=['float', 'int'])

# Plot histograms for each standardized numerical column
numerical_df.hist(figsize=(12, 8), bins=20)  # Adjust figsize and bins as needed
plt.suptitle('Histograms of Standardized Numerical Columns')
plt.show()
```



In [84]:
```
updated_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 108029 entries, 0 to 145459
Data columns (total 19 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           108029 non-null  object 
 1   Location       108029 non-null  object 
 2   MinTemp        108029 non-null  float64
 3   MaxTemp        108029 non-null  float64
 4   Rainfall       108029 non-null  float64
 5   WindGustDir    108029 non-null  object 
 6   WindGustSpeed  108029 non-null  float64
 7   WindDir9am     108029 non-null  object 
 8   WindDir3pm     108029 non-null  object 
 9   WindSpeed9am   108029 non-null  float64
 10  WindSpeed3pm   108029 non-null  float64
 11  Humidity9am    108029 non-null  float64
 12  Humidity3pm    108029 non-null  float64
 13  Pressure9am    108029 non-null  float64
 14  Pressure3pm    108029 non-null  float64
 15  Temp9am        108029 non-null  float64
 16  Temp3pm        108029 non-null  float64
 17  RainToday      108029 non-null  object 
 18  RainTomorrow   108029 non-null  object 
dtypes: float64(12), object(7)
memory usage: 16.5+ MB
```

In [85]:
```
updated_df['Date'] = pd.to_datetime(updated_df['Date'])
```

```
C:\Users\hamma\AppData\Local\Temp\ipykernel_29300\3340102702.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  updated_df['Date'] = pd.to_datetime(updated_df['Date'])
```

In [86]:
```
updated_df.head()
```

Out[86]:

	Date	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm
0	2008-12-01	Albury	0.226352	-0.148366	1.833493	W	0.619985	W	WNW
1	2008-12-02	Albury	-0.718961	0.180715	-0.379020	WNW	0.619985	NNW	WSW
2	2008-12-03	Albury	0.147576	0.270464	-0.379020	WSW	0.811509	W	WSW
3	2008-12-04	Albury	-0.435367	0.614504	-0.379020	NE	-1.295254	SE	E
4	2008-12-05	Albury	0.872317	1.257708	3.308502	W	0.332699	ENE	NW

In [87]:
```
updated_df['Year'] = updated_df['Date'].dt.year
updated_df['Month'] = updated_df['Date'].dt.month
```

```
C:\Users\hamma\AppData\Local\Temp\ipykernel_29300\3375349141.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  updated_df['Year'] = updated_df['Date'].dt.year
C:\Users\hamma\AppData\Local\Temp\ipykernel_29300\3375349141.py:2: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  updated_df['Month'] = updated_df['Date'].dt.month
```

In [88]:
```
updated_df.head()
```

Out[88]:

	Date	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm
0	2008-12-01	Albury	0.226352	-0.148366	1.833493	W	0.619985	W	WNW
1	2008-12-02	Albury	-0.718961	0.180715	-0.379020	WNW	0.619985	NNW	WSW
2	2008-12-03	Albury	0.147576	0.270464	-0.379020	WSW	0.811509	W	WSW
3	2008-12-04	Albury	-0.435367	0.614504	-0.379020	NE	-1.295254	SE	E
4	2008-12-05	Albury	0.872317	1.257708	3.308502	W	0.332699	ENE	NW

5 rows × 21 columns

In [89]:
```
#filtering out numercial columns which actually needs to have min max scaler
numeric_columns = updated_df.select_dtypes(include='number').drop(columns=['Year', 'Mon
numeric_columns
```

Out[89]:

	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Hur
0	0.226352	-0.148366	1.833493	0.619985	0.895968	0.815577	0.248334	
1	-0.718961	0.180715	-0.379020	0.619985	-1.137359	0.556465	-1.287169	
2	0.147576	0.270464	-0.379020	0.811509	0.768885	1.074688	-1.628392	
3	-0.435367	0.614504	-0.379020	-1.295254	-0.247778	-1.127759	-1.230299	
4	0.872317	1.257708	3.308502	0.332699	-0.756110	0.297354	0.873909	
...	...	...	...	...	...	...	...	
145455	-1.443701	-0.073575	-0.379020	-0.624920	0.006388	-0.868648	-0.889076	
145456	-1.317659	0.210631	-0.379020	-1.486778	0.006388	-1.127759	-0.604724	
145457	-1.034065	0.449963	-0.379020	-0.050348	-0.501944	-1.127759	-0.775335	
145458	-0.655940	0.464921	-0.379020	-0.912206	0.006388	-1.386871	-0.889076	
145459	0.462681	-0.193241	-0.379020	0.141176	0.514719	-0.091313	-0.263501	

108029 rows × 12 columns

In [90]:
```
from sklearn.preprocessing import MinMaxScaler

# Instantiate the MinMaxScaler
min_max_scaler = MinMaxScaler()

# Scale the numeric columns in updated_df
updated_df[numeric_columns.columns] = min_max_scaler.fit_transform(updated_df[numeric_c

# Display the updated DataFrame
print(updated_df)
```

```
C:\Users\hamma\AppData\Local\Temp\ipykernel_29300\1281200360.py:7: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  updated_df[numeric_columns.columns] = min_max_scaler.fit_transform(updated_df[numeri
c_columns.columns])
```

```
             Date Location   MinTemp   MaxTemp  Rainfall WindGustDir  \
0      2008-12-01   Albury  0.534626  0.493857  0.400000           W
1      2008-12-02   Albury  0.368421  0.547912  0.000000         WNW
2      2008-12-03   Albury  0.520776  0.562654  0.000000         WSW
3      2008-12-04   Albury  0.418283  0.619165  0.000000          NE
4      2008-12-05   Albury  0.648199  0.724816  0.666667           W
...           ...      ...       ...       ...       ...         ...
145455 2017-06-21    Uluru  0.240997  0.506143  0.000000           E
145456 2017-06-22    Uluru  0.263158  0.552826  0.000000         NNW
145457 2017-06-23    Uluru  0.313019  0.592138  0.000000           N
145458 2017-06-24    Uluru  0.379501  0.594595  0.000000          SE
145459 2017-06-25    Uluru  0.576177  0.486486  0.000000           W

        WindGustSpeed WindDir9am WindDir3pm  WindSpeed9am  ...  Humidity9am  \
0            0.603448          W        WNW      0.540541  ...     0.646341
1            0.603448        NNW        WSW      0.108108  ...     0.317073
2            0.637931          W        WSW      0.513514  ...     0.243902
3            0.258621         SE          E      0.297297  ...     0.329268
4            0.551724        ENE         NW      0.189189  ...     0.780488
...               ...        ...        ...           ...  ...          ...
145455       0.379310         SE        ENE      0.351351  ...     0.402439
145456       0.224138         SE          N      0.351351  ...     0.463415
145457       0.482759         SE        WNW      0.243243  ...     0.426829
145458       0.327586        SSE          N      0.351351  ...     0.402439
145459       0.517241        ESE        ESE      0.459459  ...     0.536585

        Humidity3pm  Pressure9am  Pressure3pm   Temp9am   Temp3pm  RainToday  \
0          0.212121     0.199396     0.253776  0.501370  0.509235         No
1          0.242424     0.287009     0.274924  0.509589  0.575198         No
2          0.292929     0.196375     0.302115  0.613699  0.546174         No
3          0.151515     0.498489     0.425982  0.534247  0.633245         No
4          0.323232     0.293051     0.220544  0.526027  0.717678         No
...             ...          ...          ...       ...       ...        ...
145455     0.232323     0.709970     0.652568  0.315068  0.525066         No
145456     0.202020     0.676737     0.616314  0.336986  0.580475         No
145457     0.232323     0.601208     0.546828  0.380822  0.622691         No
145458     0.232323     0.552870     0.537764  0.452055  0.620053         No
145459     0.353535     0.577039     0.580060  0.449315  0.485488         No

       RainTomorrow  Year  Month
0                No  2008     12
1                No  2008     12
2                No  2008     12
3                No  2008     12
4                No  2008     12
...             ...   ...    ...
145455           No  2017      6
145456           No  2017      6
145457           No  2017      6
145458           No  2017      6
145459           No  2017      6

[108029 rows x 21 columns]
```

In [91]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
```

In [92]:
```
Y= updated_df['RainTomorrow'].values
```

In [93]:
```
X = updated_df.drop(columns=['Date', 'RainTomorrow']).values
```

In [94]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

In [95]:
```
labelencoder = LabelEncoder()
```

In [96]:
```
updated_df.info()
```

```
 3   MaxTemp        108029 non-null  float64
 4   Rainfall       108029 non-null  float64
 5   WindGustDir    108029 non-null  object
 6   WindGustSpeed  108029 non-null  float64
 7   WindDir9am     108029 non-null  object
 8   WindDir3pm     108029 non-null  object
 9   WindSpeed9am   108029 non-null  float64
 10  WindSpeed3pm   108029 non-null  float64
 11  Humidity9am    108029 non-null  float64
 12  Humidity3pm    108029 non-null  float64
 13  Pressure9am    108029 non-null  float64
 14  Pressure3pm    108029 non-null  float64
 15  Temp9am        108029 non-null  float64
 16  Temp3pm        108029 non-null  float64
 17  RainToday      108029 non-null  object
 18  RainTomorrow   108029 non-null  object
 19  Year           108029 non-null  int32
 20  Month          108029 non-null  int32
dtypes: datetime64[ns](1), float64(12), int32(2), object(6)
memory usage: 17.3+ MB
```

In [97]:
```
X[:, 0] = labelencoder.fit_transform(X[:, 0])

X[:, 4] = labelencoder.fit_transform(X[:, 4])
X[:, 6] = labelencoder.fit_transform(X[:, 6])
X[:, 7] = labelencoder.fit_transform(X[:, 7])
X[:, 16] = labelencoder.fit_transform(X[:, 16])
```

In [98]:
```
X
```

Out[98]:
```
array([[2, 0.5346260387811634, 0.4938574938574938, ..., 0, 2008, 12],
       [2, 0.368421052631579, 0.547911547911548, ..., 0, 2008, 12],
       [2, 0.5207756232686981, 0.5626535626535626, ..., 0, 2008, 12],
       ...,
       [41, 0.31301939058171746, 0.5921375921375921, ..., 0, 2017, 6],
       [41, 0.3795013850415513, 0.5945945945945945, ..., 0, 2017, 6],
       [41, 0.5761772853185596, 0.48648648648648646, ..., 0, 2017, 6]],
      dtype=object)
```

In [99]:
```
X1 = pd.DataFrame(X)
X1
```

Out[99]:

	0	1	2	3	4	5	6	7	8	9	10	11	
0	2	0.534626	0.493857	0.4	13	0.603448	13	14	0.540541	0.615385	0.646341	0.212121	0.1
1	2	0.368421	0.547912	0.0	14	0.603448	6	15	0.108108	0.564103	0.317073	0.242424	0.2
2	2	0.520776	0.562654	0.0	15	0.637931	13	15	0.513514	0.666667	0.243902	0.292929	0.1
3	2	0.418283	0.619165	0.0	4	0.258621	9	0	0.297297	0.230769	0.329268	0.151515	0.4
4	2	0.648199	0.724816	0.666667	13	0.551724	1	7	0.189189	0.512821	0.780488	0.323232	0.2
...	...	...	...	...	...	...	...	...	...	...	...	...	
108024	41	0.240997	0.506143	0.0	0	0.37931	9	1	0.351351	0.282051	0.402439	0.232323	0.
108025	41	0.263158	0.552826	0.0	6	0.224138	9	3	0.351351	0.230769	0.463415	0.20202	0.6
108026	41	0.313019	0.592138	0.0	3	0.482759	9	14	0.243243	0.230769	0.426829	0.232323	0.6
108027	41	0.379501	0.594595	0.0	9	0.327586	10	3	0.351351	0.179487	0.402439	0.232323	0.
108028	41	0.576177	0.486486	0.0	13	0.517241	2	2	0.459459	0.435897	0.536585	0.353535	0.5

108029 rows × 19 columns

In [100]:
```
onehot_encoder = OneHotEncoder(categories='auto', sparse=False)
```

In [101]:
```
columns_to_encode = [0, 4, 6, 7, 17, 18] #we will not one hot code for rain today becau
for i in columns_to_encode:
    data = X1[i].values.reshape(-1, 1)

    # Fit and transform the selected column with one-hot encoding
    X_encoded = onehot_encoder.fit_transform(data)
    encoded_column_names = onehot_encoder.get_feature_names_out(input_features=[str(i)]

    # Convert the transformed column back to DataFrame
    encoded_df = pd.DataFrame(X_encoded, columns=encoded_column_names)

    # Drop the original column from X1
    X1.drop(columns=i, inplace=True)

    # Concatenate the encoded column to X1
    X1 = pd.concat([X1, encoded_df], axis=1)
```

```
C:\Users\hamma\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: Fut
ureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed
in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
C:\Users\hamma\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: Fut
ureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed
in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
C:\Users\hamma\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: Fut
ureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed
in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
C:\Users\hamma\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: Fut
ureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed
in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
C:\Users\hamma\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: Fut
ureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed
in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
C:\Users\hamma\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: Fut
ureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed
in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

In [102]:
```
X = X1.values
```

In [103]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
```

In [104]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state=42, test_size=0.2
```

Entropy using

In [105]:
```
clf_entropy= DecisionTreeClassifier(criterion='entropy', random_state=100,
                                    max_depth=5, min_samples_leaf=8)
```

In [106]:
```
clf_entropy.fit(X_train, Y_train)
```

Out[106]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=8,
                       random_state=100)
```

In [107]:
```
Y_pred = clf_entropy.predict(X_test)
Y_pred
```

Out[107]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

In [108]:
```
print('Accuracy is'), accuracy_score(Y_test, Y_pred)*100
```

```
Accuracy is
```

Out[108]:
```
(None, 86.73053781357031)
```

Using Gini

In [109]:
```
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=5, min_samples_split=3, m
```

In [110]:
```
clf_gini.fit(X_train, Y_train)
```

Out[110]:
```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=6, min_samples_split=3,
                       random_state=42)
```

In [111]:
```
Y_pred = clf_gini.predict(X_test)
Y_pred
```

Out[111]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

In [112]:
```
print('Accuracy is'), accuracy_score(Y_test, Y_pred)*100
```

```
Accuracy is
```

Out[112]:
```
(None, 87.02212348421735)
```

In [113]:
```
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

```
[[17979   438]
 [ 2366   823]]
```

In [114]:
```
matrix = classification_report(Y_test, Y_pred)
```

In [115]:
```
print(matrix)
```

```
              precision    recall  f1-score   support

          No       0.88      0.98      0.93     18417
         Yes       0.65      0.26      0.37      3189

    accuracy                           0.87     21606
   macro avg       0.77      0.62      0.65     21606
weighted avg       0.85      0.87      0.85     21606
```

In [ ]:
```

```