Introduction

Objective

The ultimate aim of this project is to predict if it will rain tomorrow or not. The dataset was put together by kaggle user Joe Young (https://www.kaggle.com/jsphyg (https://www.kaggle.com/jsphyg)). Actual data was acquired from number of weather stations and is published by Australian Government - Bureau of Meteorology and is available at Australian Government - Bureau of Meteorology and live as well as historical data can be accessed from http://www.bom.gov.au/climate/data (http://www.bom.gov.au/climate/data). The entire project in itself, is divided into two phases. In phase-I (Current report) we have tried to preprocess the data and explore relationships among features. phase-II will emphasize on diffrent modelling techinques and make comparisions among them. This report presents data preprocessing and exploration tasks in incremental steps, starting off by preparing the data in section 2 & exploring relationsships between features in one dimensional(univariate) and two dimensional(bivariate) spaces.
Report is compiled by utilizing awesome functionalities provided by 'Jupyter Notebook' https://jupyter.org/. Report contains textual content as well as 'Python 3.6' code to carry out data pre-processing and exploratory activities.

Data Sets

The dataset weatherAUS.csv is acquired from kaggle. It contains 142193 instances and 24 features. out of 24 features, 1 is Target feature and 23 descriptive features. However as per the guidelines of dataset provider, we have to exclude 'RISK_MM' feature from the set of descriptive features. 'RISK_MM' gives direct information about target feature and using this feature to train the model will leak the value of target feature to model. Hence, training the model using this feature will give a mirage of high accuracy during testing. So we are left with a set of 22 descriptive features and 1 target feature, based on which we will fit different binary classification models and strike a comparision among their performance in next phase of the project.

Target Feature

The response feature is RainTomorrow which is a categorical variable with 2 levels, yes or no. Yes suggesting it will rain tomorrow and no meaning otherwise. The target feature has two classes and hence it is a binary classification problem. To reiterate, The goal is to predict **whether It will rain tomorrow or not**.

Descriptive Features

The dataset contains set of ordinal, nominal and continuous features that describe weather condtion of the day prior to the day for which we are going to make prediction :

- **`date`** : *Ordinal-categorical* - Date of observation
- **`Location`** : *Nominal-categorical* - Location of weather station, **levels -** 'Canberra', 'Sydney', 'Perth', 'Darwin', 'Hobart', 'Brisbane', 'Adelaide', 'Bendigo', 'Townsville', 'AliceSprings', 'MountGambier', 'Ballarat', 'Launceston', 'Albany', 'Albury', 'MelbourneAirport', 'PerthAirport', 'Mildura', 'SydneyAirport', 'Nuriootpa', 'Sale', 'Watsonia', 'Tuggeranong', 'Portland', 'Woomera', 'Cobar', 'Cairns', 'Wollongong', 'GoldCoast', 'WaggaWagga', 'Penrith', 'NorfolkIsland', 'Newcastle', 'SalmonGums', 'CoffsHarbour', 'Witchcliffe', 'Richmond', 'Dartmoor', 'NorahHead', 'BadgerysCreek', 'MountGinini', 'Moree', 'Walpole', 'PearceRAAF', 'Williamtown', 'Melbourne', 'Nhil', 'Katherine', 'Uluru'
- **`MinTemp`** : *Continuous (in °C)* - Minimum temperature observed for that day
- **`MaxTemp`** : *Continuous (in °C)* - Maximum temperature observed for that day
- **`Rainfall`** : *Continuous (in mm)* - Amount of rainfall measured for that day
- **`Evaporation`** : *Continuous (in mm)* - Class A pan evaporation for 24 hours till 9am.
- **`Sunshine`** : *Continuous (in hours, decimal)* - No. of hours bright sunshine for the day
- **`WindGustDir`** : *Nominal-categorical* - Direction of strongest wind gust in 24 hours to midnight, **levels -** 'W', 'SE', 'E', 'N', 'SSE', 'S', 'WSW', 'SW', 'SSW', 'WNW', 'NW', 'ENE', 'ESE', 'NE', 'NNW', 'NNE'
- **`WindGustSpeed`** : *Continuous (in km/h)* - Speed of strongest wind gust in 24 hours to midnight
- **`WindDir9am`** : *Nominal-categorical* - Direction of wind gust at 9 am, **levels -** *(Same as that of feature 'WindGustDir')*
- **`WindSpeed9am`** : *Continuous (in km/h)* - Speed of wind gust averaged from 10 minutes prior to 9 am
- **`WindDir3pm`** : *Nominal-categorical* - Direction of wind gust at 3 pm, **levels -** *(Same as that of feature 'WindGustDir')*
- **`WindSpeed3pm`** : *Continuous (in km/h)* - Speed of wind gust averaged from 10 minutes prior to 3 pm
- **`WindDir3pm`** : *Nominal-categorical* - Direction of wind gust at 3 pm, **levels -** *(Same as that of feature 'WindGustDir')*
- **`Humidity9am`** : *Continuous (in %)* - Humidity % at 9 am

- **Humidity3pm** : *Continuous (in %)* - Humidity % at 3 pm
- **Pressure9am** : *Continuous (in hpa)* - Atmospheric pressure reduced to mean sea level measured at 9 am
- **Pressure3pm** : *Continuous (in hpa)* - Atmospheric pressure reduced to mean sea level measured at 3 pm
- **Cloud9am** : *Ordinal - Categorical (in Oktas)* - Fraction of sky that is overcasted by clouds in eights at 9 am *(how many eights of the sky was overcasted by clouds)*
- **Cloud3pm** : *Ordinal - Categorical (in Oktas)* - Fraction of sky that is overcasted by clouds in eights at 3 pm
- **Temp9am** : *Continuous (in °C)* - Temperature measured at 9 am
- **Temp3pm** : *Continuous (in °C)* - Temperature measured at 3 pm
- **RainToday** : *Ordinal-categorical* - Weather it rained today or not

Data Pro-processsing

Preliminaries

We get the data in form of a zip file from https://www.kaggle.com/jsphyg/weather-dataset-rattle-package/downloads/weather-dataset-rattle-package.zip/2 (https://www.kaggle.com/jsphyg/weather-dataset-rattle-package/downloads/weather-dataset-rattle-package.zip/2). we load the data into a dataframe 'weatherAus'.

In [27]:
```
# To display output of all the statements and not only last statement
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Packages
from IPython.display import display, HTML
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn import preprocessing
```

In [28]:
```
filePath = "weatherAus.csv"
Data = pd.read_csv(filePath)

print("Confirming proper data loading")
print("Loaded Data has " + str(Data.shape[0]) +" instances & " + str(Data.shape[1]) + " features")
Data.head()
```

```
Confirming proper data loading
Loaded Data has 142193 instances & 24 features
```

Out[28]:

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

5 rows × 24 columns

In [29]:
```
#Droping the unnecessary feature 'RISK_MM'

weatherAus = Data.drop(['RISK_MM', 'Date'], axis = 1)

print("Confirming feature drop")
print("")
print("weatherAus has " + str(weatherAus.shape[0]) +" instances & " + str(weatherAus.shape[1]) + " features")
```

```
Confirming feature drop

weatherAus has 142193 instances & 22 features
```

Data Cleaning and Transformation

Data type check and casting

First, we confirmed that the feature types matched the description as outlined in the documentation. It is observable that categorical features are casted to a generic object type.

To make the further operations more memory and time efficient, we will explicitly type cast these categorical features to 'category' data type.

In [30]:
```
typesBefore = weatherAus.dtypes
indices = list(typesBefore.index)
weatherAus[weatherAus.select_dtypes(['object']).columns] = weatherAus.select_d
types(['object']).apply(lambda x: x.astype('category'))

k = 0
print(f"Data Types are: ")
typesAfter = weatherAus.dtypes
print("\t  Original type >>> Converted Type")
for i,j in zip(typesBefore, typesAfter):
    print ('%-15s%-12s%-12s' % (indices[k],str(i), str(j)))
    #print( indices[k] +""+ str(i) + " >>> " + str(j))
    k += 1
```

```
Data Types are:
          Original type >>> Converted Type
Location       object      category
MinTemp        float64     float64
MaxTemp        float64     float64
Rainfall       float64     float64
Evaporation    float64     float64
Sunshine       float64     float64
WindGustDir    object      category
WindGustSpeed  float64     float64
WindDir9am     object      category
WindDir3pm     object      category
WindSpeed9am   float64     float64
WindSpeed3pm   float64     float64
Humidity9am    float64     float64
Humidity3pm    float64     float64
Pressure9am    float64     float64
Pressure3pm    float64     float64
Cloud9am       float64     float64
Cloud3pm       float64     float64
Temp9am        float64     float64
Temp3pm        float64     float64
RainToday      object      category
RainTomorrow   object      category
```

Null values checks

Following table represents feature-wise number and % of null values in the datas

In [31]:
```
print("Missing value check")
features = list(weatherAus.columns)
nullCounts = weatherAus.isnull().sum()
nanStats = pd.DataFrame()


for i in range(0,len(features)):
    nanStats = nanStats.append(pd.Series([features[i], np.int64(nullCounts[i
]), round(nullCounts[i]*100/142193,2)]), ignore_index = True)
nanStats.columns = ["Feature","Number of NaN values","% of NaN values"]

print(nanStats)
```

```
Missing value check
          Feature  Number of NaN values  % of NaN values
0        Location                   0.0             0.00
1         MinTemp                 637.0             0.45
2         MaxTemp                 322.0             0.23
3        Rainfall                1406.0             0.99
4     Evaporation               60843.0            42.79
5        Sunshine               67816.0            47.69
6     WindGustDir                9330.0             6.56
7   WindGustSpeed                9270.0             6.52
8      WindDir9am               10013.0             7.04
9      WindDir3pm                3778.0             2.66
10   WindSpeed9am                1348.0             0.95
11   WindSpeed3pm                2630.0             1.85
12    Humidity9am                1774.0             1.25
13    Humidity3pm                3610.0             2.54
14    Pressure9am               14014.0             9.86
15    Pressure3pm               13981.0             9.83
16       Cloud9am               53657.0            37.74
17       Cloud3pm               57094.0            40.15
18        Temp9am                 904.0             0.64
19        Temp3pm                2726.0             1.92
20      RainToday                1406.0             0.99
21   RainTomorrow                   0.0             0.00
```

Table shows % of null values in each feature. We will have to handle null values by either removing them from dataset or by imputing them with some other value. We can use central tendency measures like mean, median or mode to impute the null values in the features. We can also fit regression lines to predict the most likely values for that particular instance. However, we cannot generalise a single approach for all features.

Let us analyse the features having significant proportions of null values. 'Sunshine', 'Evaporation', 'Cloud3pm', 'Cloud9am' have more than 35% presence of null values.

- If we choose to impute the null values with central tendancy measure like mean or mode, we will be introducing artificial clustering in the dateset.
- If we choose to impute the null values with regression line, we will be introducing artificial linearity in the dataset.

  Both of these imputation can potentially affect the ultimate binary classification model we will be fitting in phase-II. Considering the consequences, for now we will drop these features altogether, which will be an appealing case of reducing dimensionality.

In [32]:
```
# Dropping the features with more than 35% null values
dropFeatures = ['Sunshine', 'Evaporation', 'Cloud3pm', 'Cloud9am']
for feature in dropFeatures:
    weatherAus = weatherAus.drop(feature, axis = 1)
print("Loaded Data has " + str(weatherAus.shape[0]) +" instances & " + str(wea
therAus.shape[1]) + " features")
print(str(len(dropFeatures)) + " features dropped")
```

```
Loaded Data has 142193 instances & 18 features
4 features dropped
```

For rest of the features, we can subdivide the operation in two distinct categories, namely Categorical Imputation & Continuous Imputation.

1. Categorical Null value Imputation

There are 4 categorical features with null values present among them, 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday'. In case of categorical values, there are 3 methods by which we can deal with null values.

- Drop instances with null values
- Replace null values with mode value for the feature

Feature 'RainToday' being a binary feature and having around 0.99% null values, it will be fairly practical to either drop the null instance or replace it with median values. Dropping the instances with null values will have small Information Loss which can be very well recovred from large number of instances in the dataset. Here however, we will replace the null values with median value for feature, as it will have minimal impact on the ultimate model fitting we will do.

In [33]:
```
categoricalFeaturesModeImputation = ['RainToday']

#'WindGustDir', 'WindDir9am', 'WindDir3pm'
for feature in categoricalFeaturesModeImputation:
    weatherAus[feature] = weatherAus[feature].replace(np.nan, weatherAus[featu
re].mode()[0])

print("Null values in 'RainToday' = " + str(weatherAus['RainToday'].isnull().s
um()))
```

```
Null values in 'RainToday' = 0
```

For rest of the categorical features ('WindGustDir', 'WindDir9am', 'WindDir3pm') however, we need to perform some impact analysis for each of the aforementioned methods of dealing with null values.

Let's first examine the amount of combined information lost if we drop the instaces having null values in either of 3 features. Quick calculation suggests that we will loose around 12.32% of overall information if we directly drop the instances with null values in either 3 of the features.

Let us examine effect of alternative approach of handling the null values in these features, by imputation with mode.

In [34]:
```
categoricalFeatures = ['WindGustDir', 'WindDir9am', 'WindDir3pm']
tempDataFrame = weatherAus[categoricalFeatures]
shapeBefore = tempDataFrame.shape
#print(shapeBefore)
tempDataFrame = tempDataFrame.dropna()
shapeAfter = tempDataFrame.shape
#print(shapeAfter)
print('% of instances lost after dropping the null values = ' + str(round((sha
peBefore[0] - shapeAfter[0]) * 100 /shapeBefore[0], 2)) + "%")
```

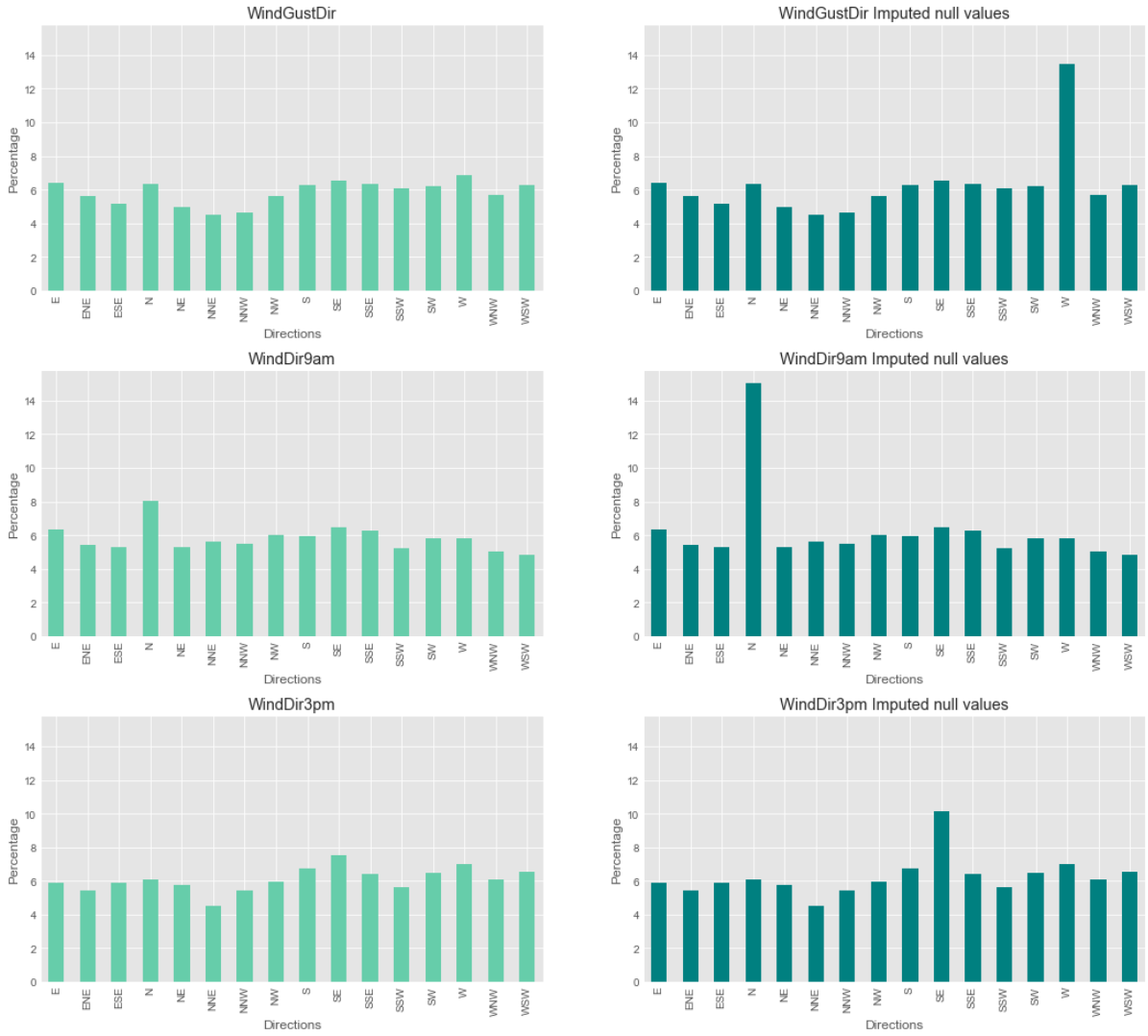
```
% of instances lost after dropping the null values = 12.32%
```

Now let us compare the effect on distribution of values if we replace values with mode value.

The graphs below represent the distribution of frequency (proportion) among features before and after imputing nulls with feature mode. Now it can be clearly seen that replacing the nulls with mode distorts the frequency distribution in a not so subtle way. This kind of distortion can affect the model accuracy.

Hence, it will be best if we drop the instances with null values in three features as large number of instances in dataset will somewhat compensate the information loss.

In [35]:
```
categoricalFeatures = ['WindGustDir', 'WindDir9am', 'WindDir3pm']

i = 1
j = 0
plt.style.use('ggplot')
fig, axs = plt.subplots(3,2, figsize=(18, 16))
fig.subplots_adjust(hspace=0.3, wspace=0.2)
for feature in categoricalFeatures:
    ax = plt.subplot(3, 2, i);
    ax.set_ylabel("Percentage");
    ax.set_xlabel("Directions");
    ax.set_ylim(0,15.8);
    ax.set_title(categoricalFeatures[j]);
    ax = (weatherAus[feature].value_counts()* 100 / 142193).sort_index().plot(
kind = 'bar', color='mediumaquamarine');
    i += 1
    ax = plt.subplot(3, 2, i);
    ax =  (weatherAus[feature].replace(np.nan, weatherAus[feature].mode()[0]).
value_counts()* 100 / 142193).sort_index().plot(kind = 'bar', color='teal');
    ax.set_ylabel("Percentage");
    ax.set_xlabel("Directions");
    ax.set_ylim(0,15.8);
    ax.set_title(categoricalFeatures[j] + " Imputed null values");
    i += 1
    j += 1
plt.show();
```



Comparing both approaches, it becomes clear that dropping the null instances will be the best way forward, as impact of information loss can be recovered from because of large number of instances in dataset. While the impact of distorted frequency distribution for 3 feature is not that easy to recover from and will affect the end model much more gravely.

Let's drop the instances with null values in either of 3 features.

In [36]:
```
categoricalFeatures = ['WindGustDir', 'WindDir9am', 'WindDir3pm']


shapeBefore = weatherAus.shape
weatherAus = weatherAus.dropna(subset = categoricalFeatures)
shapeAfter = weatherAus.shape
print("Shape change after dropping null values - " + str(shapeAfter))
print('% of instances lost after dropping the null values = ' + str(round((sha
peBefore[0] - shapeAfter[0]) * 100 /shapeBefore[0], 2)) + "%")
```

```
Shape change after dropping null values - (124668, 18)
% of instances lost after dropping the null values = 12.32%
```

2. Continuous Null value Imputation

Continuous features with null values are 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Temp9am', 'Temp3pm'.

There are 2 plausible methods to deal with the null values in these features.

- Drop instances with null values
- Replace null values with mean or median value for the feature

Considering a rather significant proportion of null values in some of the features, we will choose to replace the null values with mean values to prevent an extensive amount of information being lost.

In [37]:
```
continuousFeatures = ['MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed', 'Wind
Speed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Press
ure3pm', 'Temp9am', 'Temp3pm']

for feature in continuousFeatures:
    weatherAus[feature] = weatherAus[feature].replace(np.nan, weatherAus[featu
re].mean())

print("Total null values in entire dataset = " + str(weatherAus.isnull().sum()
.sum()))
```

```
Total null values in entire dataset = 0
```

Outliers and Sanity Checks

Examining table 1, data summary appears to be well cleaned. However, there are some obvious anamolies in the data. For example, Max value of Rainfall is 371 & with a mean value of 2.34, value is clearly an outlier. But in this scenario, outlier values, falling well away from mean will suggest anamolies in weather like storms or natural calamaties. Data in columns seems credible even when such anomalies are present. No further sanity checks are required on continuous feature at this point. anomalisitc values even though being an outlier, are of important and credible.

Table 2 describes the categorical features with their attributes. There are 49 distinct location from where observations were made. For features WindGustDir, WindDir9am, WindDir3pm which corresponds to direction of wind, there are 16 different levels, each of which corresponds to certain direction on compass, that includes 'W', 'SE', 'E', 'N', 'SSE', 'S', 'WSW', 'SW', 'SSW', 'WNW', 'NW', 'ENE', 'ESE', 'NE', 'NNW', 'NNE'. While features RainToday and RainTomorrow are binary variables with levels 'yes' or 'no'.

The reason for not explicitly handling outliers in the continuous featuers is that, outliers are rather significant instances for weather data. these outliers express the extreme weather conditions which provide significant information to our model. Hence, it is not necessary to perform any further operation at this time to clean up data.

In [38]:
```
display(HTML('<b>Table 1: Summary of continuous features</b>'))
display(weatherAus.describe(include = 'float64'))

display(HTML('<b>Table 2: Summary of categorical (object) features</b>'))
display(weatherAus.describe(include = 'category'))
```

Table 1: Summary of continuous features

	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpee
count	124668.000000	124668.000000	124668.000000	124668.000000	124668.000000	124668.0(
mean	12.417752	23.472416	2.375661	40.695263	15.022123	19.1(
std	6.361249	7.200274	8.495209	13.397250	8.315155	8.5
min	-8.500000	-4.800000	0.000000	7.000000	2.000000	2.0(
25%	7.800000	18.100000	0.000000	31.000000	9.000000	13.0(
50%	12.200000	23.000000	0.000000	39.000000	13.000000	19.0(
75%	17.000000	28.600000	0.800000	48.000000	20.000000	24.0(
max	33.900000	48.100000	367.600000	135.000000	87.000000	87.0(

Table 2: Summary of categorical (object) features

	Location	WindGustDir	WindDir9am	WindDir3pm	RainToday	RainTomorrow
count	124668	124668	124668	124668	124668	124668
unique	47	16	16	16	2	2
top	Darwin	W	N	W	No	No
freq	3119	9238	10882	9056	96912	96761

Data Exploration

We will perform Data Exploration using visualisations and will perform,

- Univariate Visualisations to understand general distribution of Data in particular features
- Multivariate Visualisations to better understand underlaying relationships and other granular inferences among features.

Univariate Visualisation

We will divide the Univariate visualisation portion in 2 parts,

- Exploration of categorical features
- Exploration of continuous features

To visualize categorical features, we will use bar plots, which is the general preference to explore data charachteristics in nominal or small scale ordinal data. Barplots reflect the proportion of observation falling into particular categories.

To visualize continuous features, we will use histogram with overlayed density plots. These plots are preferred to examine general distribution of values in feature as well as skewness of distribution.

In [39]:
```
# General graph styling for consistent layout across graphs
import seaborn as sns

plt.style.use('seaborn-white')
plt.rcParams['axes.edgecolor']='#FFFFFF'
plt.rcParams['axes.linewidth']=0
plt.rcParams['xtick.color']='#333F4B'
plt.rcParams['ytick.color']='#333F4B'
plt.rcParams['figure.figsize'] = 4.5, 3
```

Exploration of categorical features

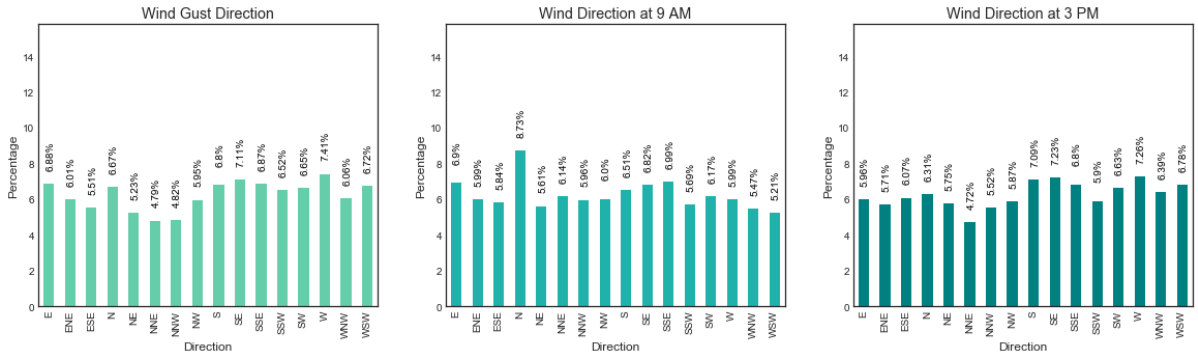
Wind Direction - 'WindGustDir', 'WindDir9am', 'WindDir3pm'

There are 3 features relating to direction of the wind flow, WindGustDir, WindDir9am & WindDir3pm. Wind Gust Direction is the direction of strongest windflow in 24 hour period. while rest of the two features are wind direction observed at 9 am and 3 pm respectively.

plotting bar plots for all 3, it becomes clear that the frequency distribution is somewhat uniform approximately within range of [-1.5% to 1.5%].

In [40]:
```
from pylab import *
plt.style.use('seaborn-white')
sns.despine(left = True)
no_of_instances = weatherAus.shape[0]
fig, axs = plt.subplots(ncols=3, figsize=(20,5))

#print(weatherAus['WindGustDir'].value_counts().sort_index() * 100 / 142193)

(weatherAus['WindGustDir'].value_counts().sort_index() * 100 / no_of_instances
).plot(kind = "bar", color='mediumaquamarine', linewidth=0, ax=axs[0])
axs[0].set_title("Wind Gust Direction")
axs[0].set_ylabel("Percentage")
axs[0].set_xlabel("Direction")
axs[0].set_ylim(0,15.8)
for i in axs[0].patches:
    # get_x pulls left or right; get_height pushes up or down
    axs[0].text(i.get_x()+.1, i.get_height()+2, \
            str(round(i.get_height(), 2))+'%', fontsize=10, rotation = 90,
                color='black')

(weatherAus['WindDir9am'].value_counts().sort_index() * 100 / no_of_instances
).plot(kind = "bar", color='lightseagreen', linewidth=0, ax=axs[1])
axs[1].set_title("Wind Direction at 9 AM")
axs[1].set_ylabel("Percentage")
axs[1].set_xlabel("Direction")
axs[1].set_ylim(0,15.8)
for i in axs[1].patches:
    # get_x pulls left or right; get_height pushes up or down
    axs[1].text(i.get_x()+.1, i.get_height()+2, \
            str(round(i.get_height(), 2))+'%', fontsize=10, rotation = 90,
                color='black')

(weatherAus['WindDir3pm'].value_counts().sort_index() * 100 / no_of_instances
).plot(kind = "bar", color='teal', linewidth=0, ax=axs[2])
axs[2].set_title("Wind Direction at 3 PM")
axs[2].set_ylim(0,15.8)
axs[2].set_ylabel("Percentage")
axs[2].set_xlabel("Direction")
for i in axs[2].patches:
    # get_x pulls left or right; get_height pushes up or down
    axs[2].text(i.get_x()+.1, i.get_height()+2, \
            str(round(i.get_height(), 2))+'%', fontsize=10, rotation = 90,
                color='black')
plt.show();
```

```
<Figure size 324x216 with 0 Axes>
```



Exploration of continuous features

We have plotted side by side plots for weather measures like temperature, windspeed, humidity and pressure, at two different times of day, 9 am and 3 pm. Individual plots show underlaying frequency distribution as well as mean of the feature. By considering adjcent plots, we can see how value of features gets affected by comparing means and frequency distributions. Comparing multiple plots we can have rough estimation on how changes in one feature will affect other feature. However, these estimation will be pretty crude at this stage of exploration, and we will explore this plausible estimation further during multivariet exploration.

Looking at plot 1-A, we can observe that temperature at 9am has mean of 17.27 °C. Distribution is bell shaped indicating a roughly normal distribution. Same kind of distribution shape is observable in graph 1-B. However, mean of temperature measurement at 3pm is 21.89 °C, which is significantly higher than mean temperature at 9am. It supports our general intiution that as mornings are colder than afternoons as sun makes a higher angle with east horizon.

Plot 2-A & 2-B indicate wind speed measured in kmph at 9am and 3pm. Average wind speed in the morning is 15.02 kmph & in the afternoon it is 19.18 kmph. Considering large number of instances on which this graphs are based, this increase in average windspeed is rather significant. || It is also intersting that both temperature & windspeed increase with time. ||

Plot 3-A & 3-B represent % Humidity in air at 9am and 3pm respectively. Average humidity decreases from around 67% at 9am to around 50% at 3pm. Distribution of values is rather widespread and is symmetric to an extent. However, in the morning (9am - 3-A plot), there is significant presence of high humidity near the end of graph, which can potentially affect mean, pulling it towards a higher % value. This spike can be because of extreme weather conditions.

last pair of plots represent a atmospheric pressure at 9am & 3pm. The spike in the center is the first noticable thing in both graphs. The spike can be a result of our data cleaning process. During preprocessing phase, we replcaed nulls present in the feature with mean of the feature. However, apart from a slight decrease in the pressure, there is no plausible relationship that we can deduce from these set of graphs.

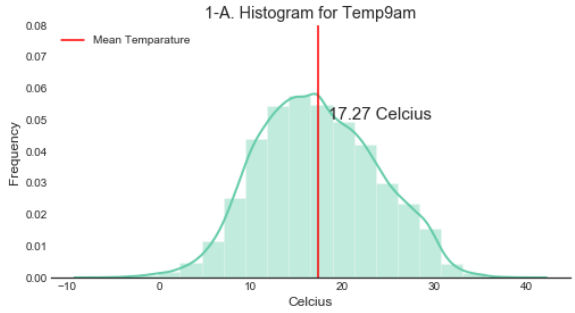
If we consider temperature & windspeed, it is plausible that higher temperature in the afternoons can partly account for increase in the mean wind speed at 3pm.

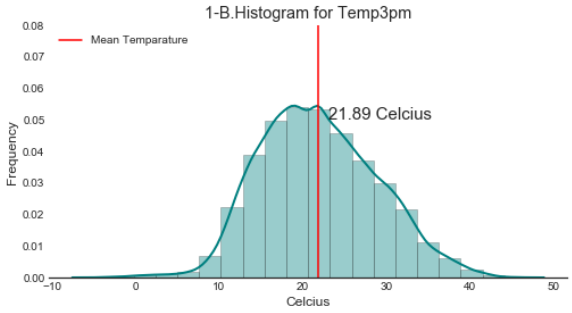
According to laws of thermodynamics, at higher temperatures air's capacity to hold water vapour increases. If additional vapour is not added in the system, overall humidity of air will decrease. So, during afternoon, when temperature rises, humidity should decrease. So theoritically, there is an inverse relationship between temperature and humidity. However, there can be numerous other feature which can affect this relationship. But at this stage, our data confirms with the plausible inversed relationship between these two features.

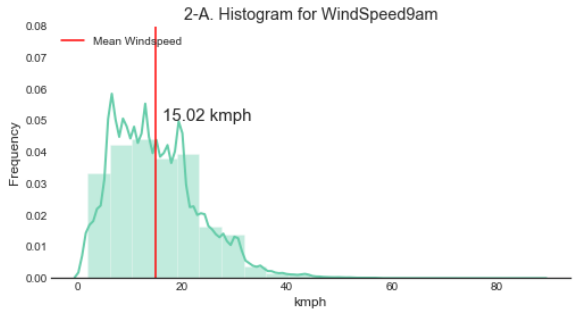
We will examine relationships among these feature in a more detailed manner in next section.

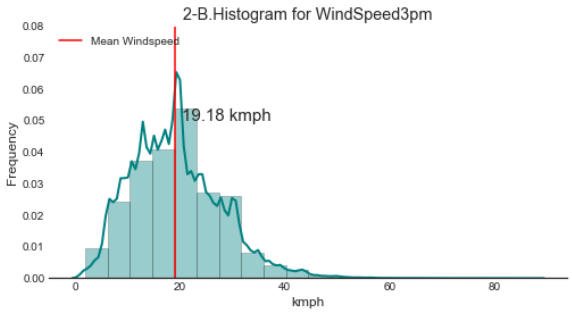
In [41]:
```
con = ['Temp9am', 'Temp3pm', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'H
umidity3pm', 'Pressure9am', 'Pressure3pm','Rainfall', 'WindGustSpeed']
plot1 = ['Temp9am', 'WindSpeed9am', 'Humidity9am', 'Pressure9am']
plot2 = ['Temp3pm', 'WindSpeed3pm', 'Humidity3pm', 'Pressure3pm']
exLabels = ["Temparature", "Windspeed", "Humidity", "Pressure"]
xLabels = ["Celcius", "kmph", "%", "Hpa"]

fig, axs = plt.subplots(nrows = 4, ncols=2, figsize=(18,20))

i=0
sns.despine(left = True)

for f1,f2 in zip(plot1,plot2):

    mean1 = round(weatherAus[f1].mean(), 2)
    mean2 = round(weatherAus[f2].mean(), 2)

    sns.distplot(weatherAus[f1], hist=True, kde=True, bins=int(20), color = 'm
ediumaquamarine', ax = axs[i][0],
             hist_kws={'edgecolor':'white'},kde_kws={'linewidth': 2})
    l11 = axs[i][0].axvline(x= mean1, label='Mean ' + exLabels[i], c='r')
    axs[i][0].text(x=  mean1 + 1.2, y = 0.05, s=str(mean1)+" "+ xLabels[i], fo
ntsize = 15 )
    axs[i][0].set_title(str(i+1)+"-A. Histogram for " + f1 )
    axs[i][0].set_ylabel("Frequency")
    axs[i][0].set_xlabel(xLabels[i])
    axs[i][0].set_ylim(0,0.08)
    axs[i][0].legend(loc = "upper left")

    sns.distplot(weatherAus[f2], hist=True, kde=True,  bins=int(20), color =
'teal', ax = axs[i][1],
             hist_kws={'edgecolor':'black'}, kde_kws={'linewidth': 2})
    l2 = axs[i][1].axvline(x= mean2, label='Mean ' + exLabels[i], c='r')
    axs[i][1].text(x= mean2 + 1.2, y = 0.05, s=str(mean2)+" "+ xLabels[i], fon
tsize = 15)
    axs[i][1].set_title(str(i+1)+"-B.Histogram for " + f2)
    axs[i][1].set_ylabel("Frequency")
    axs[i][1].set_xlabel(xLabels[i])
    axs[i][1].set_ylim(0,0.08)
    axs[i][1].legend(loc = "upper left")

    i+=1
plt.subplots_adjust(hspace = 0.3)
plt.show();
```

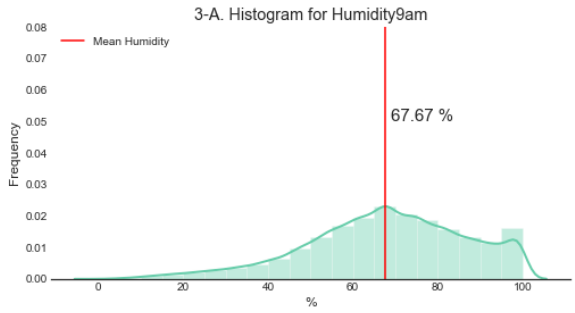
```
C:\Users\jigar\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is deprecate
d; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be in
terpreted as an array index, `arr[np.array(seq)]`, which will result either i
n an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

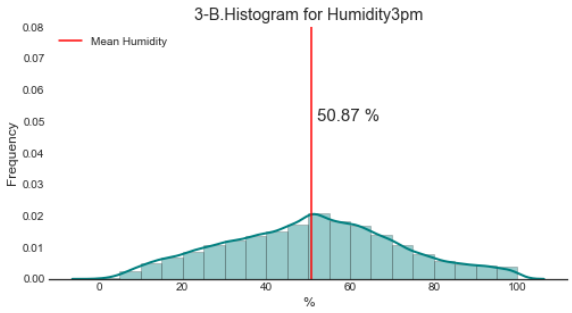


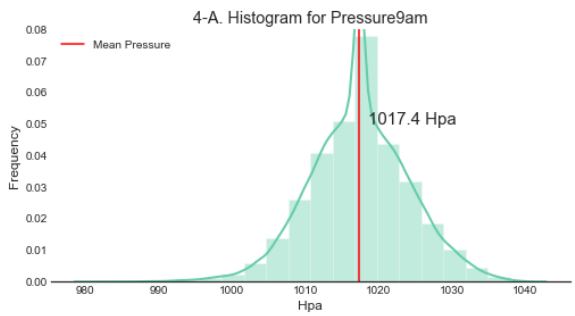


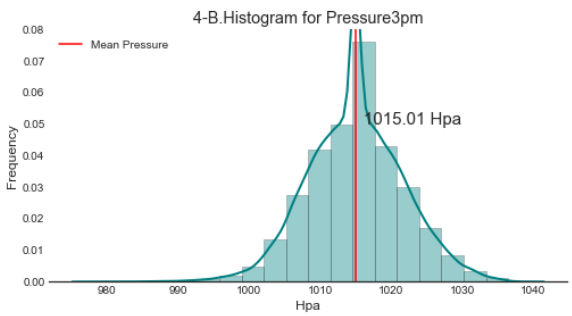






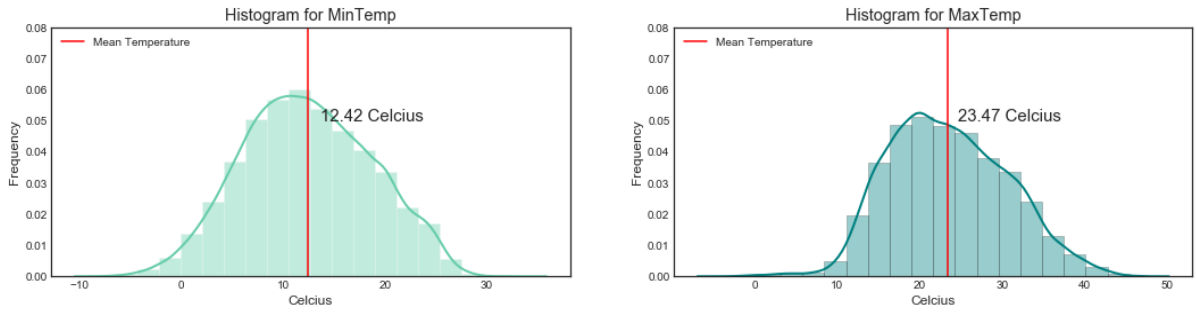






Following plots show the difference between

In [42]:
```
fig, axs = plt.subplots(ncols=2, figsize=(18,4))

f1 = 'MinTemp'
f2 = 'MaxTemp'

mean1 = round(weatherAus[f1].mean(), 2)
mean2 = round(weatherAus[f2].mean(), 2)

sns.distplot(weatherAus[f1], hist=True, kde=True, bins=int(20), color = 'mediu
maquamarine', ax = axs[0],
         hist_kws={'edgecolor':'white'},kde_kws={'linewidth': 2})
l11 = axs[0].axvline(x= mean1, label='Mean Temperature', c='r')
axs[0].text(x=  mean1 + 1.2, y = 0.05, s=str(mean1)+" Celcius", fontsize = 15
)
axs[0].set_title("Histogram for " + f1 )
axs[0].set_ylabel("Frequency")
axs[0].set_xlabel("Celcius")
axs[0].set_ylim(0,0.08)
axs[0].legend(loc = "upper left")

sns.distplot(weatherAus[f2], hist=True, kde=True,  bins=int(20), color = 'tea
l', ax = axs[1],
         hist_kws={'edgecolor':'black'}, kde_kws={'linewidth': 2})
l2 = axs[1].axvline(x= mean2, label='Mean Temperature', c='r')
axs[1].text(x= mean2 + 1.2, y = 0.05, s=str(mean2)+" Celcius", fontsize = 15)
axs[1].set_title("Histogram for " + f2)
axs[1].set_ylabel("Frequency")
axs[1].set_xlabel("Celcius")
axs[1].set_ylim(0,0.08)
axs[1].legend(loc = "upper left")

plt.show();
```

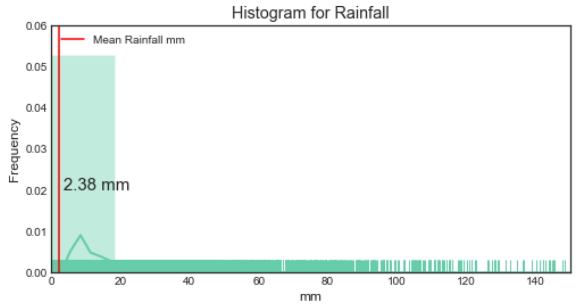


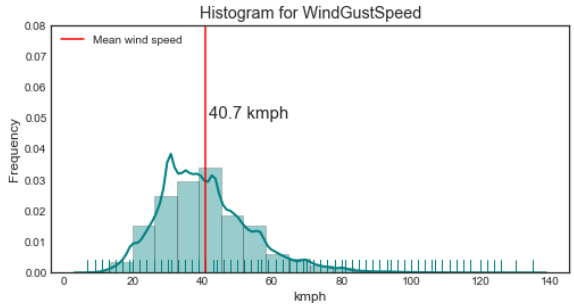
Histogram for Rainfall & WindGustSpeed

from the left histogram, it is observable that rainfall has a high concerntration of values in the first bin rangin from 0 to 20mm. with a mean of 2.38 mm and max value laying beyond 350mm, distribution can be termed as highly asymmetric. We will have to normalize this feature after confirming our findings in multivariate exploration.

Right histogram represents the frequency distribution of WindGustSpeed. Distribution appears to be right skewed and the mean is 40.7 kmph, which is significantly higher than means of Wind Speed at 9am(15.02 kmph) and 3pm(19.18 kmph).

In [43]:
```
fig, axs = plt.subplots(ncols=2, figsize=(18,4))

f1 = 'Rainfall'
f2 = 'WindGustSpeed'

mean1 = round(weatherAus[f1].mean(), 2)
mean2 = round(weatherAus[f2].mean(), 2)

sns.distplot(weatherAus[f1], hist=True, kde=True, bins=int(20), color = 'mediu
maquamarine', ax = axs[0], rug = True,
         hist_kws={'edgecolor':'white'},kde_kws={'linewidth': 2})
l11 = axs[0].axvline(x= mean1, label='Mean Rainfall mm', c='r')
axs[0].text(x=  mean1 + 1.2, y = 0.02, s=str(mean1)+" mm", fontsize = 15 )
axs[0].set_title("Histogram for " + f1 )
axs[0].set_ylabel("Frequency")
axs[0].set_xlabel("mm")
axs[0].set_ylim(0,0.06)
axs[0].set_xlim(0,150)
axs[0].legend(loc = "upper left")

sns.distplot(weatherAus[f2], hist=True, kde=True,  bins=int(20), color = 'tea
l', ax = axs[1], rug= True,
         hist_kws={'edgecolor':'black'}, kde_kws={'linewidth': 2})
l2 = axs[1].axvline(x= mean2, label='Mean wind speed', c='r')
axs[1].text(x= mean2 + 1.2, y = 0.05, s=str(mean2)+" kmph", fontsize = 15)
axs[1].set_title("Histogram for " + f2)
axs[1].set_ylabel("Frequency")
axs[1].set_xlabel("kmph")
axs[1].set_ylim(0,0.08)
axs[1].legend(loc = "upper left")

plt.show();
```





Multivariate Visualisation

Categorised scatter plots

In this section, we will inspect underlaying relationships between numerical variable pairs. Dataset contains 9am - 3pm feature pairs for measurements like Temperature, Humidity, WindSpeed & Humidity. It will be interesting to see how values of these time bound features are interrelated (How morning weather affects afternoon weather). We will try to figure out how these features are interrelated and how features like 'RainToday' and 'RainTomorrow'(Target) is affected by features like temperature, humidity and pressure. Throughout the exploration, We will make put up several plausible statements from our analysis, though some of them will make more sense than other.

In all the plots in this section, sea green points in the graph represent No - RainToday or No - RainTomorrow instances. While crimson(red) points represent Yes - RainToday or Yes - RainTomorrow instances. Size of points, which is not very distinguishable, represents the Rainfall in mm.

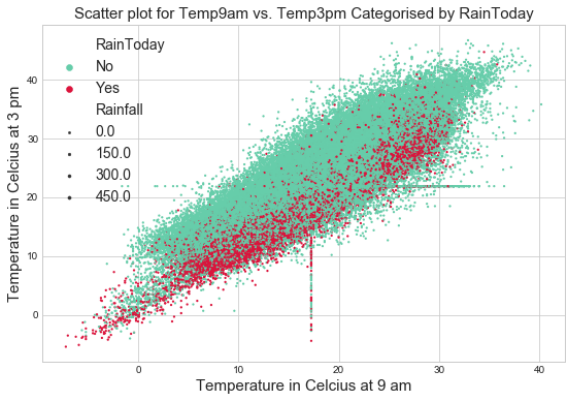
1. Temperature at 9am & Temperature at 3pm by RainToday & RainTomorrow

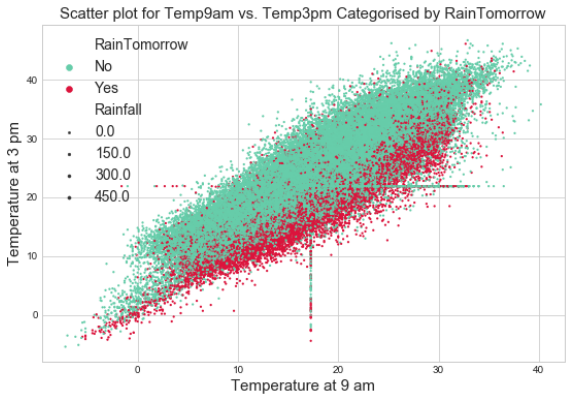
From the following graphs, several interesting observations can be stated.

- Temp9am & Temp3pm have a strong linear correlationship. Which makes sense in a way that, if morning temperature is high, afternoon temperature will potentially be high as well, and vice versa. A rather obvious correlation in most case. However, it can be other way round depending on other weather conditions.
- in the both plots, crimson points representing Yes - RainToday, are spreaded out at the bottom edge of the distribution clusters. The bottom edge of cluster represent a comparatively low 9am & 3pm Temperatures. It is plausible that days with lower morning and afternoon temperatures have higher probability of raining on the same day as well as next day than days with comparetively higher temperatures.

In [44]:
```
category = ["No","Yes"]
plt.style.use('seaborn-whitegrid')
sns.despine(left = True)
fig, axs = plt.subplots(nrows = 1, ncols=2, figsize=(20,6))
sns.scatterplot(x="Temp9am", y="Temp3pm", hue="RainToday", size="Rainfall", pa
lette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[0])
axs[0].set_title("Scatter plot for Temp9am vs. Temp3pm Categorised by RainToda
y",fontsize=15)
axs[0].set_xlabel("Temperature in Celcius at 9 am",fontsize=15)
axs[0].set_ylabel("Temperature in Celcius at 3 pm",fontsize=15)
axs[0].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Temp9am", y="Temp3pm", hue="RainTomorrow", size="Rainfall",
palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[1])
axs[1].set_title("Scatter plot for Temp9am vs. Temp3pm Categorised by RainTomo
rrow" ,fontsize=15)
axs[1].set_xlabel("Temperature at 9 am",fontsize=15)
axs[1].set_ylabel("Temperature at 3 pm",fontsize=15)
axs[1].legend(loc = "best",fontsize = 14)
plt.show();
```

```
<Figure size 324x216 with 0 Axes>
```





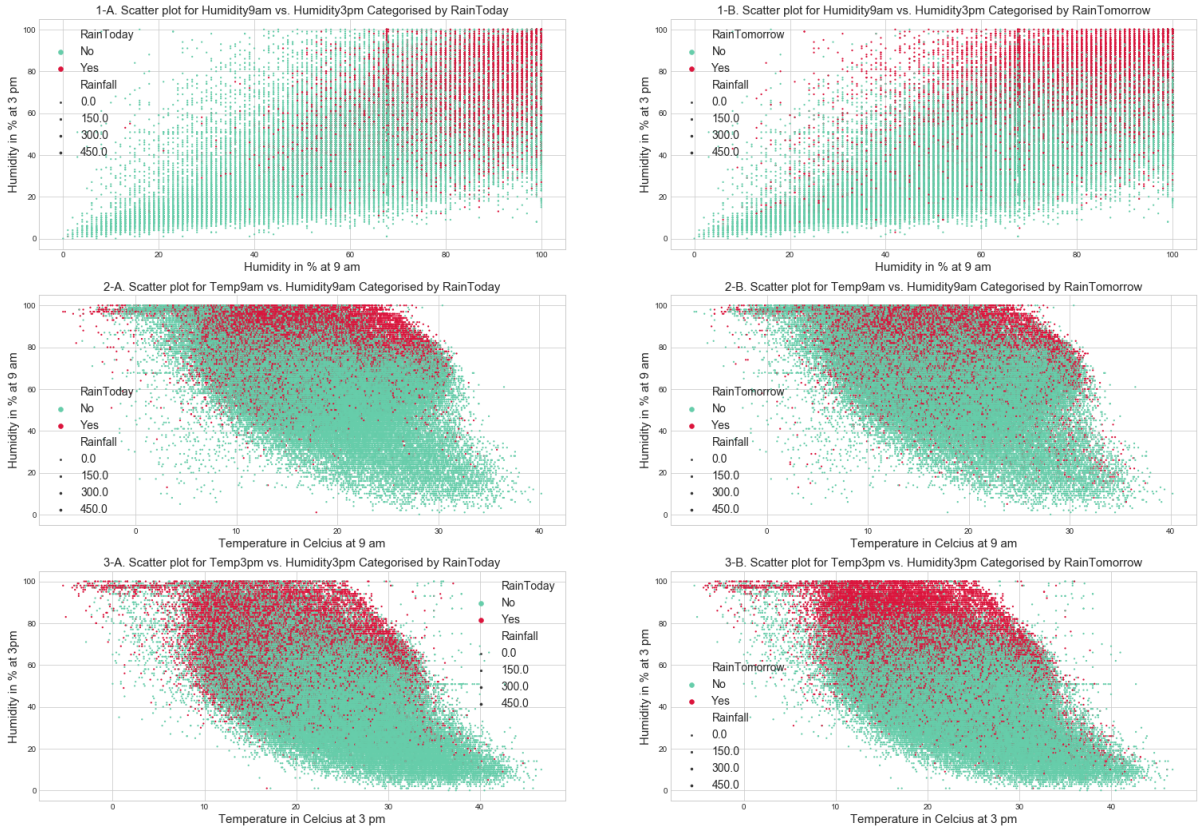
2. Humidity & Humidity vs. Temperature by RainToday & RainTomorrow

- From graphs 1-A & 1-B, it is quite clear that Humidity9am & Humidity3pm have a positive correlation. It can be explained as high humidity in morning generally results in high humidity at afternoon and vice versa. Crimson clusters at the top right corners of both 1-A & 1-B graphs suggest that, on days with high humidity percentages are potentially more likely to rain.
- By making a closer examination of crimson clusters in 1-A & 1-B, it is evident that cluster shifts bit towards top centre part of the plot. By which we can state a plausible statement, days with higher humidity % at 3pm will have more chance of raining on the following day compared to days with low humidity % at 3pm.
- From graphs in rows 2 & 3, it is rather clear that temperature and humidity have a negative correlation. Which strengthens plausible hypothesis we made in Univariate Exploration, stating negative correlation between temperature and humidity.
- It is also intersting to note the dense crimson clustering in graphs 2-A & 3-B. plot 2-A represents RainToday feature and is based on morning measurements. While plot 3-B represents RaintTomorrow feature and is based on afternoon measurements. combining these two pieces of information, we can say that afternoon measurements are provide more information regarding RainTomorrow feature than morning measurements. which seems to be true intuitively, as weather condition can change throught out the day, and hence measurements made later in the day can provide more information regarding weather of the following day.
- All 4 graphs in rows 2 & 3 suggest that days with high % of humidity has more chances of raining on the same day, and on the next day as well.

In [45]:
```
category = ["No","Yes"]
plt.style.use('seaborn-whitegrid')
sns.despine(left = True)


fig, axs = plt.subplots(nrows = 3, ncols=2, figsize=(26,18))

sns.scatterplot(x="Humidity9am", y="Humidity3pm", hue="RainToday", size="Rainf
all", palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[0][0])
axs[0][0].set_title("1-A. Scatter plot for Humidity9am vs. Humidity3pm Categor
ised by RainToday",fontsize=15)
axs[0][0].set_xlabel("Humidity in % at 9 am",fontsize=15)
axs[0][0].set_ylabel("Humidity in % at 3 pm",fontsize=15)
axs[0][0].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Humidity9am", y="Humidity3pm", hue="RainTomorrow", size="Ra
infall", palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[0][1])
axs[0][1].set_title("1-B. Scatter plot for Humidity9am vs. Humidity3pm Categor
ised by RainTomorrow",fontsize=15 )
axs[0][1].set_xlabel("Humidity in % at 9 am",fontsize=15)
axs[0][1].set_ylabel("Humidity in % at 3 pm",fontsize=15)
axs[0][1].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Temp9am", y="Humidity9am", hue="RainToday", size="Rainfall"
, palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[1][0])
axs[1][0].set_title("2-A. Scatter plot for Temp9am vs. Humidity9am Categorised
by RainToday",fontsize=15 )
axs[1][0].set_xlabel("Temperature in Celcius at 9 am",fontsize=15)
axs[1][0].set_ylabel("Humidity in % at 9 am",fontsize=15)
axs[1][0].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Temp9am", y="Humidity9am", hue="RainTomorrow", size="Rainfa
ll", palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[1][1])
axs[1][1].set_title("2-B. Scatter plot for Temp9am vs. Humidity9am Categorised
by RainTomorrow",fontsize=15 )
axs[1][1].set_xlabel("Temperature in Celcius at 9 am",fontsize=15)
axs[1][1].set_ylabel("Humidity in % at 9 am",fontsize=15)
axs[1][1].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Temp3pm", y="Humidity3pm", hue="RainToday", size="Rainfall"
, palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[2][0])
axs[2][0].set_title("3-A. Scatter plot for Temp3pm vs. Humidity3pm Categorised
by RainToday",fontsize=15 )
axs[2][0].set_xlabel("Temperature in Celcius at 3 pm",fontsize=15)
axs[2][0].set_ylabel("Humidity in % at 3pm",fontsize=15)
axs[2][0].legend(loc = "best",fontsize = 14)
```

```
sns.scatterplot(x="Temp3pm", y="Humidity3pm", hue="RainTomorrow", size="Rainfa
ll", palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[2][1])
axs[2][1].set_title("3-B. Scatter plot for Temp3pm vs. Humidity3pm Categorised
by RainTomorrow",fontsize=15 )
axs[2][1].set_xlabel("Temperature in Celcius at 3 pm",fontsize=15)
axs[2][1].set_ylabel("Humidity in % at 3 pm",fontsize=15)
axs[2][1].legend(loc = "best",fontsize = 14)

plt.show();
```

```
<Figure size 324x216 with 0 Axes>
```



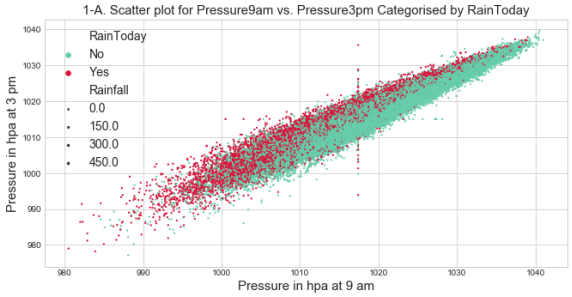
3. Pressure & Pressure vs. Temperature by RainToday & RainTomorrow

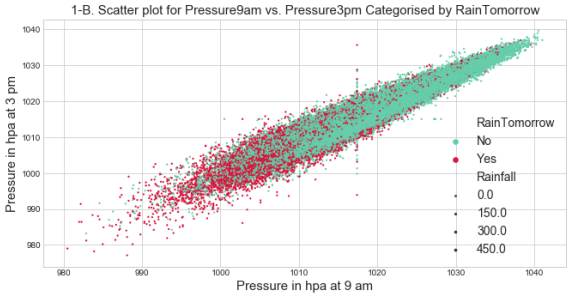
- From plots 1-A & 1-B a strong linear postive correlation can be deduces between features Pressure9am & Pressure3pm. major crimson clusters are located at bottom left corners of graphs. that indicate comparatively lower pressure increases probability of rain.
- From graphs in rows 2 & 3, it is observable that temperature and pressure have a negative correaltion. Increase in one causes reduction in the other.
- Very simillar types of crimson clusters are observable in the graph 2-A, 2-B, 3-A & 3-B. All the clusters are in the lower left corner of graphs. From that we can state that lower pressure - lower temperature situations have a comperatively higher probability of rain.

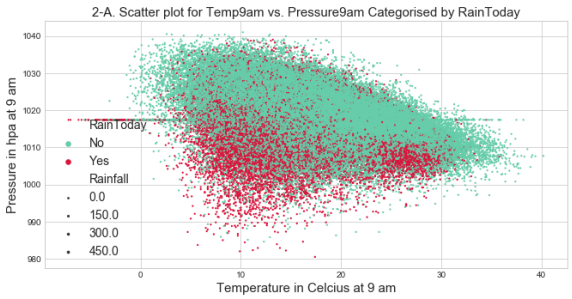
In [46]:
```
category = ["No","Yes"]
plt.style.use('seaborn-whitegrid')
sns.despine(left = True)


fig, axs = plt.subplots(nrows = 3, ncols=2, figsize=(24,18))

sns.scatterplot(x="Pressure9am", y="Pressure3pm", hue="RainToday", size="Rainf
all", palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[0][0])
axs[0][0].set_title("1-A. Scatter plot for Pressure9am vs. Pressure3pm Categor
ised by RainToday",fontsize=15 )
axs[0][0].set_xlabel("Pressure in hpa at 9 am",fontsize=15)
axs[0][0].set_ylabel("Pressure in hpa at 3 pm",fontsize=15)
axs[0][0].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Pressure9am", y="Pressure3pm", hue="RainTomorrow", size="Ra
infall", palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[0][1])
axs[0][1].set_title("1-B. Scatter plot for Pressure9am vs. Pressure3pm Categor
ised by RainTomorrow",fontsize=15 )
axs[0][1].set_xlabel("Pressure in hpa at 9 am",fontsize=15)
axs[0][1].set_ylabel("Pressure in hpa at 3 pm",fontsize=15)
axs[0][1].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Temp9am", y="Pressure9am", hue="RainToday", size="Rainfall"
, palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[1][0])
axs[1][0].set_title("2-A. Scatter plot for Temp9am vs. Pressure9am Categorised
by RainToday",fontsize=15 )
axs[1][0].set_xlabel("Temperature in Celcius at 9 am",fontsize=15)
axs[1][0].set_ylabel("Pressure in hpa at 9 am",fontsize=15)
axs[1][0].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Temp9am", y="Pressure9am", hue="RainTomorrow", size="Rainfa
ll", palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[1][1])
axs[1][1].set_title("2-B. Scatter plot for Temp9am vs. Pressure9am Categorised
by RainTomorrow",fontsize=15 )
axs[1][1].set_xlabel("Temperature in Celcius at 9 am",fontsize=15)
axs[1][1].set_ylabel("Pressure in hpa at 9 am",fontsize=15)
axs[1][1].legend(loc = "best",fontsize = 14)

sns.scatterplot(x="Temp3pm", y="Pressure3pm", hue="RainToday", size="Rainfall"
, palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[2][0])
axs[2][0].set_title("3-A. Scatter plot for Temp3pm vs. Pressure3pm Categorised
by RainToday",fontsize=15)
axs[2][0].set_xlabel("Temperature in Celcius at 3 pm",fontsize=15)
axs[2][0].set_ylabel("Pressure in hpa at 3pm",fontsize=15)
axs[2][0].legend(loc = "best",fontsize = 14)
```

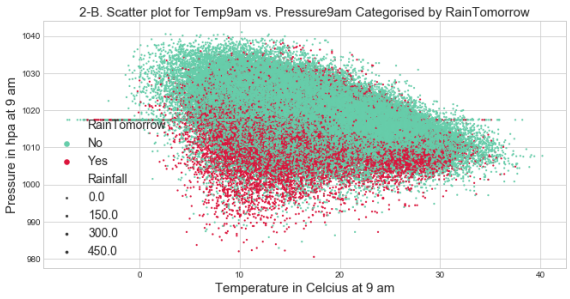
```
sns.scatterplot(x="Temp3pm", y="Pressure3pm", hue="RainTomorrow", size="Rainfa
ll", palette=["mediumaquamarine","crimson"],
                hue_order=category, sizes=(5, 10), linewidth=0, data=weatherAu
s,ax = axs[2][1])
axs[2][1].set_title("3-B. Scatter plot for Temp3pm vs. Pressure3pm Categorised
by RainTomorrow",fontsize=15 )
axs[2][1].set_xlabel("Temperature in Celcius at 3 pm",fontsize=15)
axs[2][1].set_ylabel("Pressure in hpa at 3 pm")
axs[2][1].legend(loc = "best",fontsize = 14)

plt.show();
```

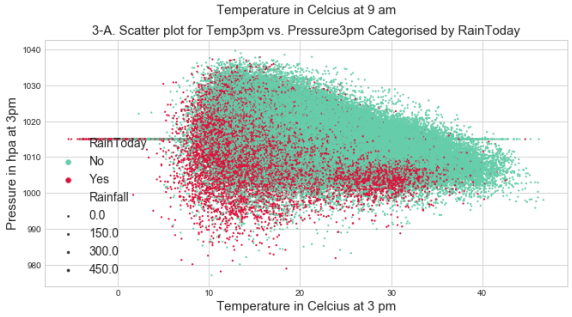
```
<Figure size 324x216 with 0 Axes>
```

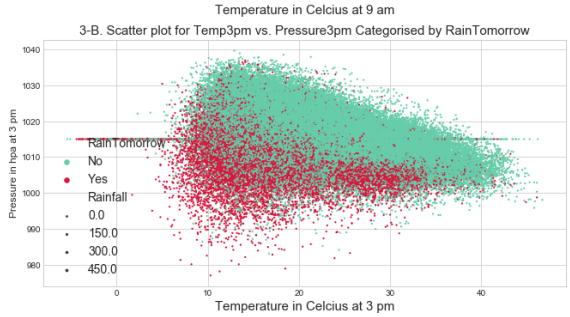












The scatter plots above bring forth some interesting relationships among the features. Some of them can be used during modeling phase to improve model performance. Linearity between paired features (9am - 3pm pairs) is significant and should be considering while fitting models to this dataset.

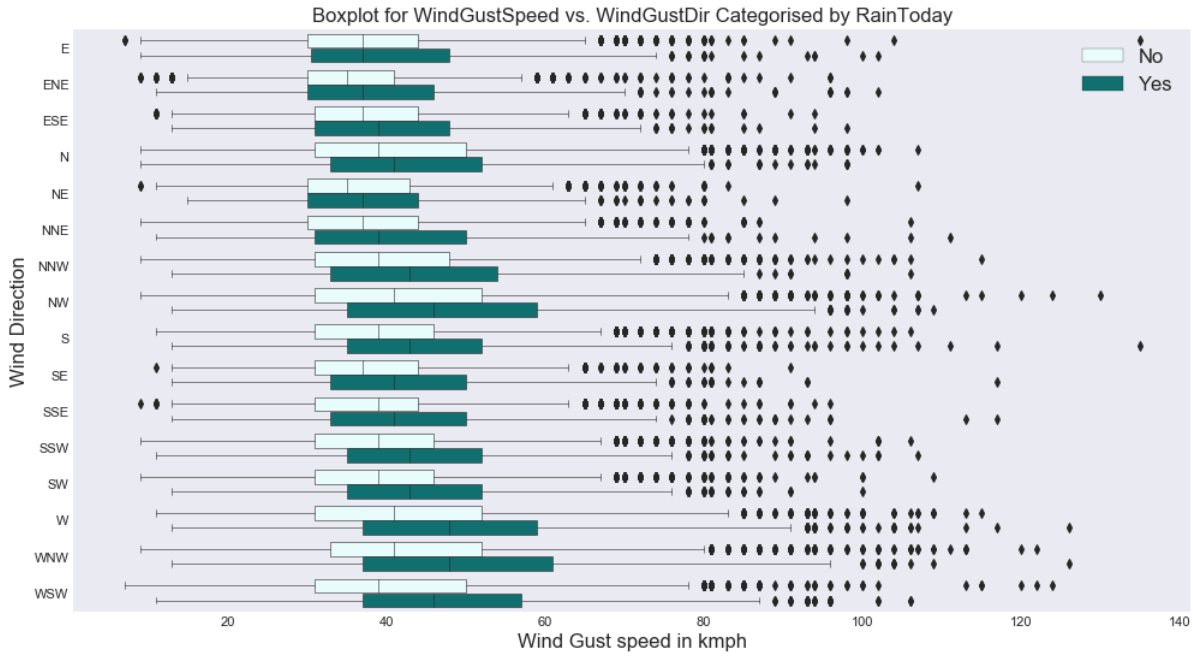
Categorical vs Continuous - WindGustDir vs WindGustSpeed subcategorised by RainToday

The boxplot represents WindGustDir vs WindGustSpeed. It is futher subcategorised by categorical feature 'RainToday'.

Comparing different directions, it becomes clear that W, NW, NNW, WNW directions have strong winds gusts. It is also interesting that all the RainToday - Yes boxes have higher average windspeed and IQR than that of corresponding RainToday - No boxes. Presence of numerous outliers represent higher than normal windspeed from that particular direction.

Apart from that however, the graphs provides very less information about features.

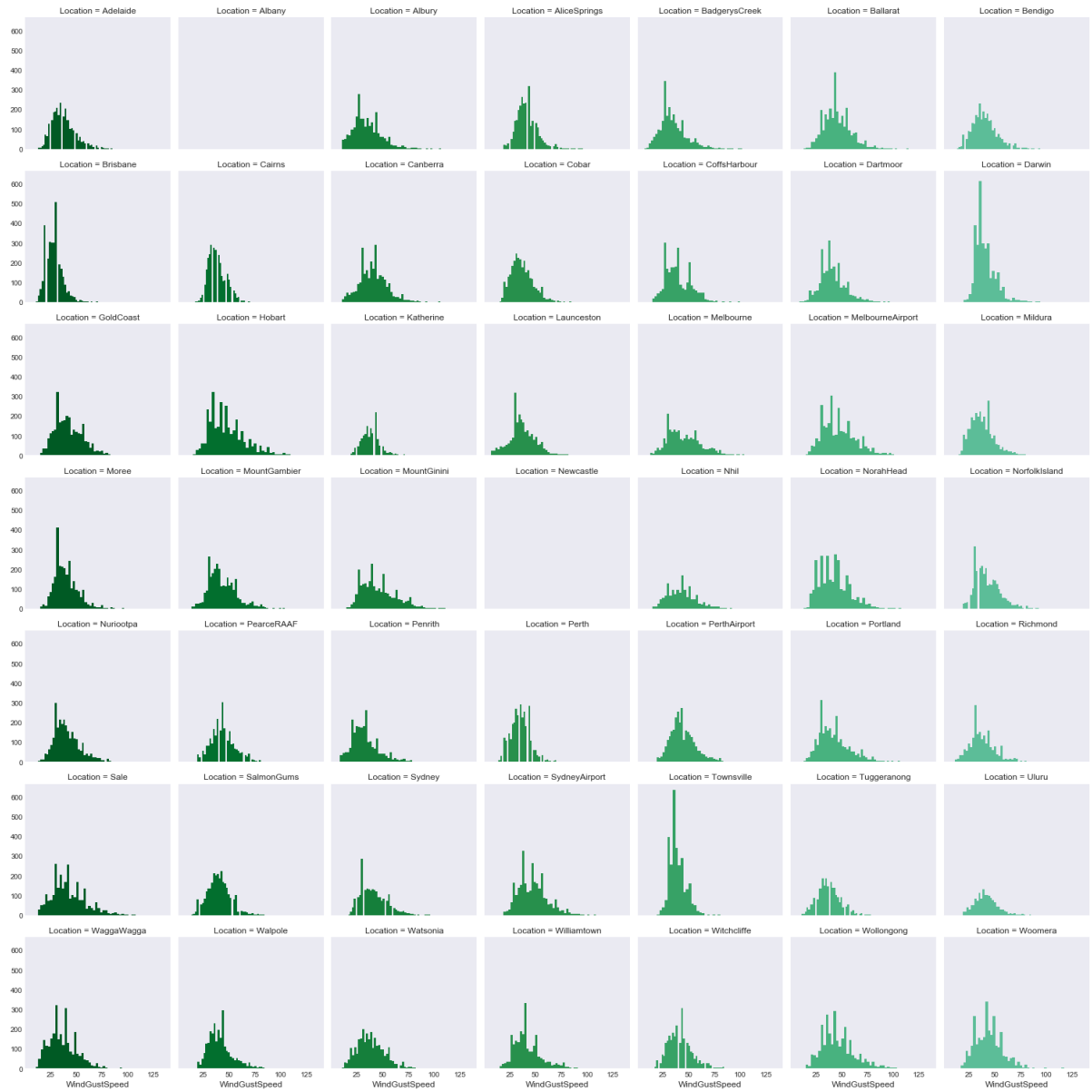
In [47]:
```
figure(num=None, figsize=(15, 8), dpi=80, facecolor='w', edgecolor='k')
plt.style.use("seaborn-dark")
sns.boxplot(x="WindGustSpeed", y="WindGustDir",hue='RainToday', data=weatherAu
s, color = "teal", linewidth = 0.5)
plt.title("Boxplot for WindGustSpeed vs. WindGustDir Categorised by RainToday"
,fontsize=15 )
plt.ylabel("Wind Direction",fontsize=15)
plt.xlabel("Wind Gust speed in kmph",fontsize=15)
plt.legend(loc="upper right", fontsize = 15)
plt.show();
```



FacetGrid of WindGustSpeed by Location

- We can observe the distribution of WindGustSpeed by Location. Data appears to be missing for Locations "Albany" & "NewCastle". Most of the data is right skewed. Some locations like "Midura" & "GoldCoast" have high spikes in the distribution which can potentially induce unnecessary bias in the model. But due to high number of instances it will be rectified. Hence we don't need to individually inspect or impute this values.

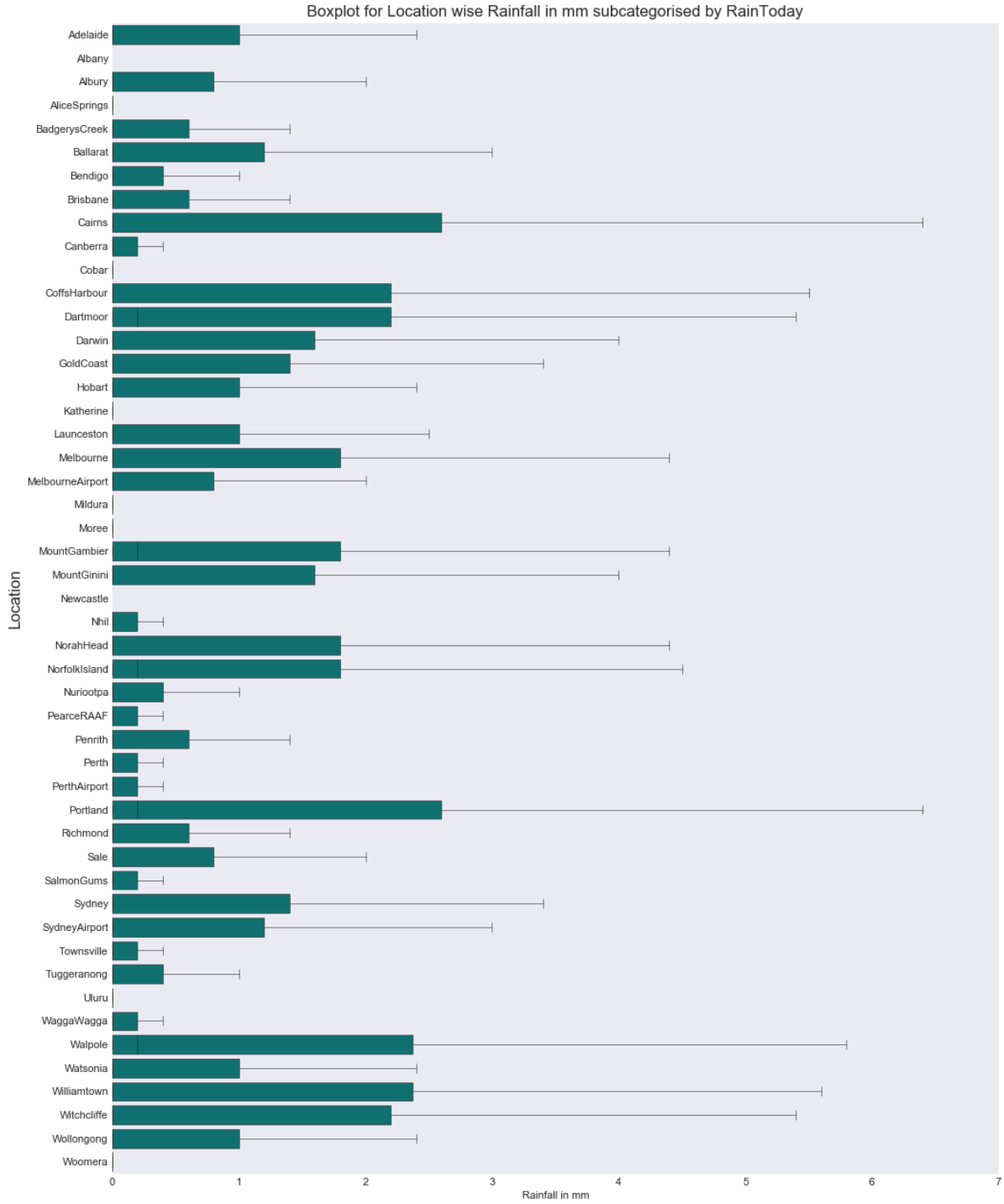
In [48]:
```
grid = sns.FacetGrid(weatherAus[['WindGustSpeed','Location']], col="Location",
hue="Location", palette=(sns.color_palette("BuGn_r",14))[:7],
                     col_wrap=7, height=3, legend_out=True, despine = True)
grid.map(plt.hist, "WindGustSpeed", bins = 45)
plt.show();
```



Boxplot for Locationwise Rainfall in mm (outlier supressed)

boxplot represents the majority Rainfall measurements in mm. It is evident that most of the cities Q3 number below 3mm. Outliers for this plot have been suppressed as plotting all the outliers will result in overplotting and render the graph unreadable. around 75% values for all Locations being in range of (0,3), it will be important to normalise this variable, so it doesn't deteriorate the model accuracy because of outliers.

In [49]:
```
figure(num=None, figsize=(15, 20), dpi=80, facecolor='w', edgecolor='k')
plt.style.use("seaborn-dark")
sns.boxplot(x="Rainfall", y="Location", data=weatherAus, color = "teal", linew
idth = 0.5, showfliers = False)
plt.title("Boxplot for Location wise Rainfall in mm subcategorised by RainToda
y",fontsize=15 )
plt.ylabel("Location",fontsize=15)
plt.xlabel("Rainfall in mm",fontsize=10)
plt.xlim(0,7)

plt.show();
```



We will normalize the feature Rainfall to convert the wide range of values into a normalised range. We have used Min Max Normalisation. Data is now in the range of 0 to 1 with a std of 0.02311 & mean of 0.006463.

In [50]:
```
weatherAus["Rainfall"] = (weatherAus["Rainfall"] - weatherAus["Rainfall"].min
()) / (weatherAus["Rainfall"].max() - weatherAus["Rainfall"].min())
print(weatherAus['Rainfall'].describe())
```

```
count    124668.000000
mean          0.006463
std           0.023110
min           0.000000
25%           0.000000
50%           0.000000
75%           0.002176
max           1.000000
Name: Rainfall, dtype: float64
```

Summary

In Phase 1, We have imputed Null values in 'RainToday' feature with mode and in other continuous feature with mean. We have dropped instances with nulls in either 'WindGustDir','WindDir9am','WindDir3pm' features. We have not removed or imputed outliers due to their statistical significance. We have dropped "Date" and 'Risk_MM' features as they were not required for purpose of this project. We have dropped "Cloud9am", "Cloud3pm", "Sunshine" & "Evaporation" features because of high (>35%) presence of nulls among them. From exploration section we found out that 9am & 3pm feature pairs are strongly correlated and we will have to consider this corelation while fitting models. We also normalised feature 'Rainfall' using MinMax normalisation. we discovered how features are distributed and how this distribution serves to our target feature 'RainTomorrow'.

Significant Continuous Features - "MinTemp", "MaxTemp", "Temp9am", "Temp3pm", "Pressure9am", "Pressure3pm", "Humidity9am", "Humidity3pm", "WindSpeed9am", "WindSpeed3pm", "WindGustSpeed", "Rainfall" Significant Categorical Features - "WindGustDir", "WindDir9am", "WindDir3pm", "RainToday"