RLadies
等雨停

Waiting Is Not Easy.

Olly

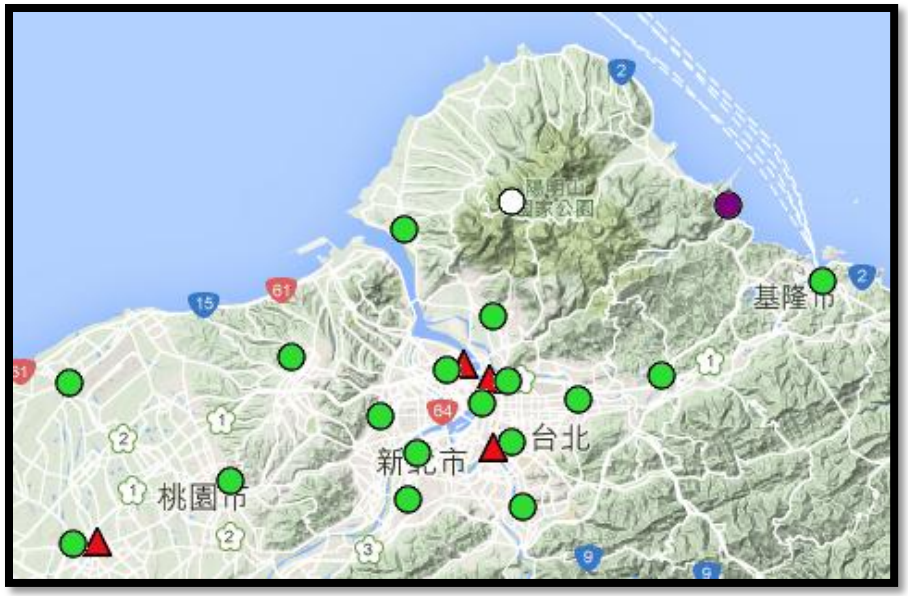
Olly

- Data Mining Programmer/Analyst
- Data Science Enthusiast
- Mining for…
  - Email Marketing
  - Subscribers behavior prediction
  - Fraud Prevention
- Bachelor & Master's Degree of Public Health, Taipei Medical University

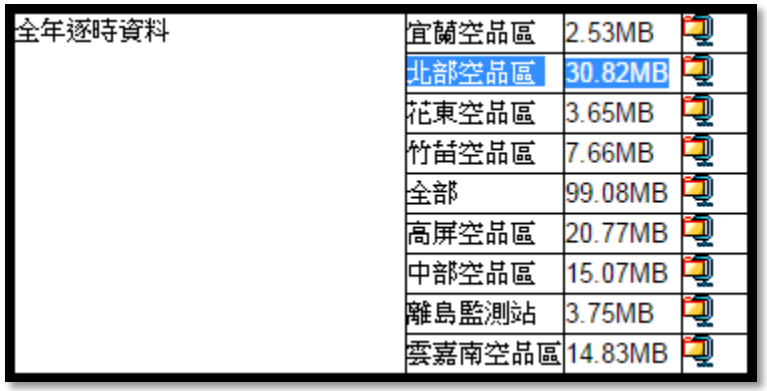
Weather Podcast

- 行政院環境保護署空氣品質監測網資料
  - http://taqm.epa.gov.tw/taqm/tw/YearlyDataDownload.aspx

- 古亭站



Source Data Download

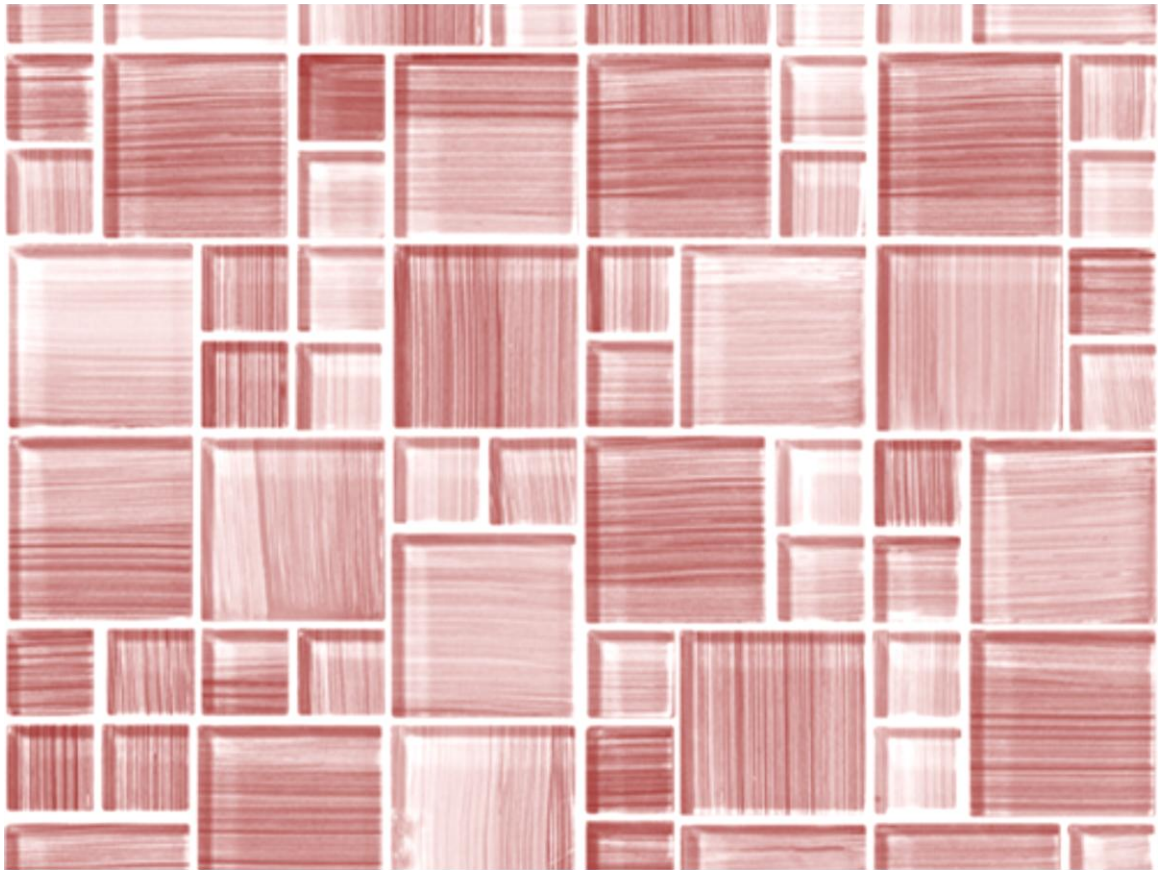


Name	Unit	Description
AMB_TEMP	℃	大氣溫度
RAINFALL	mm	雨量
RH	%	相對溼度
WIND_SPEED	m/sec	風速(以每小時最後10分鐘算術平均)
WIND_DIREC	degress	風向(以每小時最後10分鐘向量平均)
WS_HR	m/sec	風速小時值(以整個小時算術平均)
WD_HR	degress	風向小時值(以整個小時向量平均)

日期	測站	測項	00	01	02	03
2015/01/01	古亭	AMB_TEMP	16	16	16	15
2015/01/01	古亭	RAINFALL	NR	NR	NR	NR
2015/01/01	古亭	RH	60	61	61	62
2015/01/01	古亭	WD_HR	70	82	79	77
2015/01/01	古亭	WIND_DIREC	73	81	86	82
2015/01/01	古亭	WIND_SPEED	2.9	3.3	2.7	2
2015/01/01	古亭	WS_HR	3.1	2.7	2.6	2.2
2015/01/02	古亭	AMB_TEMP	13	13	13	13
2015/01/02	古亭	RAINFALL	NR	NR	NR	NR
2015/01/02	古亭	RH	62	61	62	62
2015/01/02	古亭	WD_HR	99	102	105	89
2015/01/02	古亭	WIND_DIREC	101	102	99	89
2015/01/02	古亭	WIND_SPEED	2.6	2.8	2.4	2.4
2015/01/02	古亭	WS_HR	2.4	2.8	2.6	2

Data duration : 2013/01/01 – 2015/12/31

- Data clean
- Data transformation
- Outlier detection
- Data split
  - Training (90%)
  - Test(10%)
- Training
- Testing
- Model performance



Let’s code!!

Import modules

```
import pandas as pd
import numpy as np
import scipy.stats as stats
import pylab as pl
from sklearn.cross_validation import train_test_split
from patsy import dmatrices
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
```

Import raw data

In [3]:
```
# Read excel
xlsx102 = pd.ExcelFile(r'C:\Users\Olly\Desktop\Rain\102年古亭站_20140417.xls')
xlsx103 = pd.ExcelFile(r'C:\Users\Olly\Desktop\Rain\103年古亭站_20150324.xls')
xlsx104 = pd.ExcelFile(r'C:\Users\Olly\Desktop\Rain\104年古亭站_20160318.xls')

sheet1 = pd.concat([xlsx102.parse(0),xlsx103.parse(0),xlsx104.parse(0)])
print("★☆ Rows : ", sheet1.shape[0], "; Columns : ", sheet1.shape[1])
sheet1.index= range(sheet1.shape[0])
sheet1
```

19673	2015/12/30	古亭	PM2.5	14	12	15	17	16	12	4	...	0	4	6	8	7	3	4
19674	2015/12/30	古亭	RAINFALL	NR	NR	NR	NR	NR	NR	NR	...	NR	NR	NR	NR	NR	NR	NR
19675	2015/12/30	古亭	RH	91	93	92	93	92	93	92	...	93	93	93	94	94	94	93
19676	2015/12/30	古亭	SO2	1.6	2.5	3.1	4.2	2.7	1.5	1.4	...	4.2	2.9	2.7	1.9	2.4	1.9	2.9
19677	2015/12/30	古亭	THC	1.9	1.9	2	2	2	2	2	...	2	2	2	2.1	2.1	2.1	2

Indexes introduce

In [4]:
```
date = pd.Series(sheet1.iloc[:,0].real).unique()
index = pd.Series(['AMB_TEMP','RAINFALL', 'RH', 'WIND_SPEED', 'WIND_DIREC', 'WS_HR', 'WD_HR','PM10','PM2.5'])

# Data Strаucture
print("★☆ Data Period :", date[:1],"~", date[-1:])
print()
print("★☆ All indexes : \n",
      index[0],"   : 大氣溫度(℃)\n",
      index[1],"   : 雨量(mm)\n",
      index[2],"         : 相對溼度(%)\n",
      index[3]," : 風速(m/sec)\n",
      index[4]," : 風向(degress)\n",
      index[5],"      : 風速小時值(m/sec)\n",
      index[6],"      : 風向小時值(degress)\n",
      index[7],"       : 懸浮微粒(μg/m3)\n",
      index[8],"      : 細懸浮微粒(μg/m3)\n",
     )
```

```
★☆ Data Period : ['2013/01/01'] ~ ['2015/12/31']

★☆ All indexes :
 AMB_TEMP    : 大氣溫度(℃)
 RAINFALL    : 雨量(mm)
 RH          : 相對溼度(%)
 WIND_SPEED  : 風速(m/sec)
 WIND_DIREC  : 風向(degress)
 WS_HR       : 風速小時值(m/sec)
 WD_HR       : 風向小時值(degress)
 PM10        : 懸浮微粒(μg/m3)
 PM2.5       : 細懸浮微粒(μg/m3)
```

Data transpose

In [23]:
```
def DataTranspose(indexInput):
    dd_OK = pd.DataFrame()
    a = sheet1[sheet1['測項']==indexInput]
    a.drop(['測站','測項','日期'], axis=1, inplace=True)
    b = a.T
    col = list(b.columns)
    for c in col:
        d = b[c]
        dd_OK = pd.concat([dd_OK,d])
    dd_OK.columns=[indexInput]
    return dd_OK

AMB_TEMP = DataTranspose('AMB_TEMP')
RAINFALL = DataTranspose('RAINFALL')
RH = DataTranspose('RH')
WIND_SPEED = DataTranspose('WIND_SPEED')
WIND_DIREC = DataTranspose('WIND_DIREC')
WS_HR = DataTranspose('WS_HR')
WD_HR = DataTranspose('WD_HR')
PM10 = DataTranspose('PM10')
PM25=DataTranspose('PM2.5')

stra = pd.concat([AMB_TEMP, RAINFALL, RH, WIND_SPEED, WIND_DIREC, WS_HR, WD_HR, PM10, PM25], axis=1 )
stra["Hour"] = stra.index
stra.index = range(stra.shape[0])

print("★☆ After transpose :")
print("    Rows : ", stra.shape[0], "; Columns : ", stra.shape[1])
print("★☆ First 5 rows↓")
stra.head(5)
```

```
★☆ After transpose :
    Rows :  26280 ; Columns :  10
★☆ First 5 rows↓
```

Out[23]:

	AMB_TEMP	RAINFALL	RH	WIND_SPEED	WIND_DIREC	WS_HR	WD_HR	PM10	PM2.5	Hour
0	13	NR	71	2.5	98	2.3	97	53	27	00
1	13	NR	72	2.6	76	2.6	93	54	27	01
2	14	NR	71	2.6	69	2.6	69	62	25	02
3	14	NR	69	2	83	2.1	76	55	27	03
4	14	NR	70	1.8	100	1.9	79	50	28	04

Chage 'Date' to 'Season' & change 'Hour' to 'DayPeriod'

In [24]:
```
# Add Date
Date_col=[]
for d in date:
    eachday = [d]*24
    Date_col = Date_col +eachday
stra["Date"] = Date_col

# Add Season
def DateToSeason(row):
    if (row['Date'][5:7] == '03')|(row['Date'][5:7] == '04')|(row['Date'][5:7] == '05'):
        return 'Q1'
    elif (row['Date'][5:7] == '06')|(row['Date'][5:7] == '07')|(row['Date'][5:7] == '08'):
        return 'Q2'
    elif (row['Date'][5:7] == '09')|(row['Date'][5:7] == '10')|(row['Date'][5:7] == '11'):
        return 'Q3'
    elif (row['Date'][5:7] == '12')|(row['Date'][5:7] == '01')|(row['Date'][5:7] == '02'):
        return 'Q4'
    else:
        return np.nan

stra['Season'] = stra.apply(DateToSeason, axis=1)
del stra['Date']

#pd.Series(stra.iloc[:,9].real).unique()

# Change hour to DayPeriod
def Hour_To_DayPeriod(row):
    if ((row['Hour'] == '00') | (row['Hour'] == '01')| (row['Hour'] == '02')| (row['Hour'] == '03')| (row['Hour'] == '04')| (row[
        return 'Midnight'
    elif ((row['Hour'] == '06') | (row['Hour'] == '07')| (row['Hour'] == '08')| (row['Hour'] == '09')| (row['Hour'] == 10)| (row[
        return 'Morning'
    elif ((row['Hour'] == 12) | (row['Hour'] == 13)| (row['Hour'] == 14)| (row['Hour'] == 15)| (row['Hour'] == 16)| (row['Hour'] 
        return 'Afternoon'
    elif ((row['Hour'] == 18) | (row['Hour'] == 19)| (row['Hour'] == 20)| (row['Hour'] == 21)| (row['Hour'] == 22)| (row['Hour']
        return 'Evening'
    else:
        return np.nan

stra['DayPeriod'] = stra.apply(Hour_To_DayPeriod, axis=1)

del stra['Hour']

#pd.crosstab(stra['DayPeriod'],stra['Hour'])
print("★☆ After chage 'Date' to 'Season' & change 'hour' to 'DayPeriod' :")
print("    Rows : ", stra.shape[0], "; Columns : ", stra.shape[1])
print("★☆ First 5 rows↓")
stra.head(5)
```

```
★☆ After chage 'Date' to 'Season' & change 'hour' to 'DayPeriod' :
    Rows :  26280 ; Columns :  11
★☆ First 5 rows↓
```

Out[24]:

	AMB_TEMP	RAINFALL	RH	WIND_SPEED	WIND_DIREC	WS_HR	WD_HR	PM10	PM2.5	Season	DayPeriod
0	13	NR	71	2.5	98	2.3	97	53	27	Q4	Midnight
1	13	NR	72	2.6	76	2.6	93	54	27	Q4	Midnight
2	14	NR	71	2.6	69	2.6	69	62	25	Q4	Midnight

Dealing Missing Values

In [25]:
```
print("★☆ Dealing Missing Values")
print("    # 表示儀器檢核為無效值\n"
      "    * 表示程式檢核為無效值\n"
      "    x 表示人工檢核為無效值\n"
      "    NR 表示無降雨\n"
      "    空白 表示缺值")
print()
mis=[]
for s in range(0,9):
    ss = list(pd.Series(stra.iloc[:,s].real).unique())
    for sss in ss:
       mis.append(sss)

miss=[]
for el in mis:
        if type(el) == type('XX'):
            miss.append(el)
print("★☆ Total ",len(miss)," types of missing value :\n ", miss)
print()
print("★☆ Replace missing to Nan; replace NR to 0")
DealNR = stra.replace('NR', 0)
rep = DealNR.replace(miss, np.nan)
rep[:5]
```

```
★☆ Dealing Missing Values
    # 表示儀器檢核為無效值
    * 表示程式檢核為無效值
    x 表示人工檢核為無效值
    NR 表示無降雨
    空白 表示缺值
```

Create Target variable

In [26]:
```
print("★☆ Create & Mapping Target variable")
a = rep
b = list(a['RAINFALL'])
b.pop(0)
b.append(np.nan)
a['RAINFALL_Next1'] = b

c = list(a['RAINFALL'])
c.pop(0)
c.pop(0)
c.append(np.nan)
c.append(np.nan)
a['RAINFALL_Next2'] = c

print("  'RAINFALL'       : ", len(list(a['RAINFALL'])), "Rows; ",list(a['RAINFALL'])[:5], list(a['RAINFALL'])[-5:])
print("  'RAINFALL_Next1' : ", len(b), "Rows; ", b[:5], b[-5:])
print("  'RAINFALL_Next2' : ", len(c), "Rows; ", c[:5], c[-5:])

def TargetTransform(row):
    if ((row['RAINFALL'] > 0) & (row['RAINFALL_Next1'] > 0) & (row['RAINFALL_Next2'] == 0)) :
        return 1
    elif ((row['RAINFALL'] > 0) & (row['RAINFALL_Next1'] > 0) & (row['RAINFALL_Next2'] > 0)) :
        return 0
    else:
        return np.nan

a['Target'] = a.apply(TargetTransform, axis=1)
```

```
★☆ Create & Mapping Target variable
  'RAINFALL'       :  26280 Rows;  [0.0, 0.0, 0.0, 0.0, 0.0] [0.0, 0.0, 0.0, 0.0, 0.0]
  'RAINFALL_Next1' :  26280 Rows;  [0.0, 0.0, 0.0, 0.0, 0.0] [0.0, 0.0, 0.0, 0.0, nan]
  'RAINFALL_Next2' :  26280 Rows;  [0.0, 0.0, 0.0, 0.0, 0.0] [0.0, 0.0, 0.0, nan, nan]
```

In [27]:
```
#a.sort_values(['Target_RAINFALL'], ascending=[0])
TT = a[(a['Target']>=0)]
print("★☆ Keep valueable data ('Target'= 0 OR 1)")
print("    Rows : ", TT.shape[0], "; Columns : ", TT.shape[1])
print()
print("★☆ 'Nan'% in each variable :")
print(TT.isnull().sum()/int(TT.shape[0])*100)
print()
print("★☆ After Clean 'Nan' :")
for c in index:
      TT = TT[np.isfinite(TT[c])]
print("    Rows : ", TT.shape[0], "; Columns : ", TT.shape[1])
print()
print("★☆ First 5 rows↓")
TT[:5]
```

```
★☆ Keep valueable data ('Target'= 0 OR 1)
    Rows :  2163 ; Columns :  14

★☆ 'Nan'% in each variable :
AMB_TEMP           0.046232
RAINFALL           0.000000
RH                 0.462321
WIND_SPEED         0.000000
WIND_DIREC         0.000000
WS_HR              0.000000
WD_HR              0.046232
PM10               0.647249
PM2.5              2.820157
Season             0.000000
DayPeriod          0.000000
RAINFALL_Next1     0.000000
RAINFALL_Next2     0.000000
Target             0.000000
dtype: float64

★☆ After Clean 'Nan' :
    Rows :  2085 ; Columns :  14

★☆ First 5 rows↓
```

Out[27]:

	AMB_TEMP	RAINFALL	RH	WIND_SPEED	WIND_DIREC	WS_HR	WD_HR	PM10	PM2.5	Season	DayPeriod	RAINFALL_Next1	RAINF
9	13.0	0.4	85.0	1.9	88.0	2.0	74.0	52.0	35.0	Q4	Morning	0.2	1.4
10	13.0	0.2	90.0	1.9	87.0	1.8	78.0	46.0	37.0	Q4	Morning	1.4	0.8
11	13.0	1.4	90.0	2.8	84.0	2.4	86.0	40.0	34.0	Q4	Morning	0.8	0.4
12	13.0	0.8	91.0	2.6	106.0	2.1	89.0	42.0	19.0	Q4	Afternoon	0.4	0.4
13	13.0	0.4	91.0	2.5	87.0	2.2	88.0	40.0	23.0	Q4	Afternoon	0.4	0.8

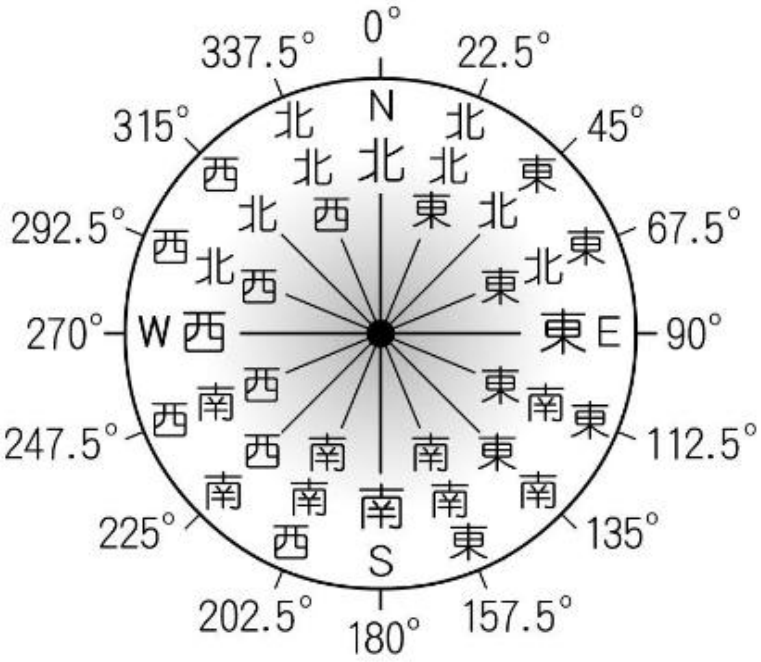
Add varibles (WS_Change, WD_Change, PM10_25)

In [28]:
```
del TT['RAINFALL_Next1']
del TT['RAINFALL_Next2']
TT['WS_Change'] = TT['WIND_SPEED'] - TT['WS_HR']
del TT['WIND_SPEED']
TT['WD_Change'] = abs(TT['WIND_DIREC'] - TT['WD_HR'])
del TT['WIND_DIREC']
TT['PM10_25'] = TT['PM10'] - TT['PM2.5']
TT['PM25']=TT['PM2.5']
del TT['PM10']
del TT['PM2.5']
TT.head()
```

Out[28]:

	AMB_TEMP	RAINFALL	RH	WS_HR	WD_HR	Season	DayPeriod	Target	WS_Change	WD_Change	PM10_25	PM25
9	13.0	0.4	85.0	2.0	74.0	Q4	Morning	0.0	-0.1	14.0	17.0	35.0
10	13.0	0.2	90.0	1.8	78.0	Q4	Morning	0.0	0.1	9.0	9.0	37.0
11	13.0	1.4	90.0	2.4	86.0	Q4	Morning	0.0	0.4	2.0	6.0	34.0
12	13.0	0.8	91.0	2.1	89.0	Q4	Afternoon	0.0	0.5	17.0	23.0	19.0
13	13.0	0.4	91.0	2.2	88.0	Q4	Afternoon	0.0	0.3	1.0	17.0	23.0

Wind direction transform (WD_HR)



In [29]:
```
def WD_HR_To_WD_HR_gp(row):
    if (row['WD_HR'] >0) & (row['WD_HR'] <=45):
        return 'EN'
    elif (row['WD_HR'] >45) & (row['WD_HR'] <=90):
        return 'EN'
    elif (row['WD_HR'] >90) & (row['WD_HR'] <=135):
        return 'ESE'
    elif (row['WD_HR'] >135) & (row['WD_HR'] <=180):
        return 'SSE'
    elif (row['WD_HR'] >180) & (row['WD_HR'] <=225):
        return 'SSW'
    elif (row['WD_HR'] >225) & (row['WD_HR'] <=270):
        return 'WSW'
    elif (row['WD_HR'] >270) & (row['WD_HR'] <=315):
        return 'WN'
    elif (row['WD_HR'] >315):
        return 'WN'
    else:
        return np.nan

TT['WD_HR_gp'] = TT.apply(WD_HR_To_WD_HR_gp, axis=1)
del TT['WD_HR']

print("★☆ Final Data snapshot")
TT.head()
```

```
★☆ Final Data snapshot
```

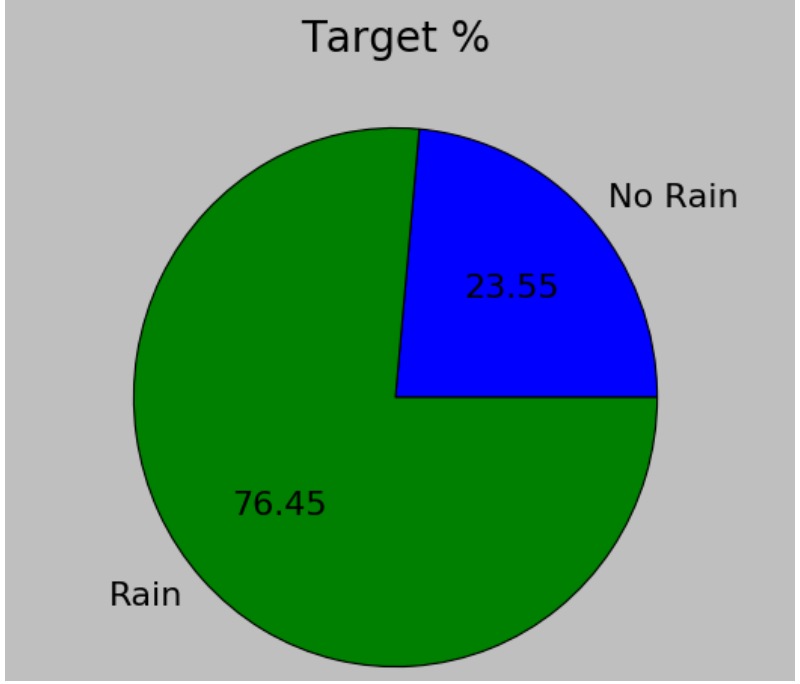
Out[29]:

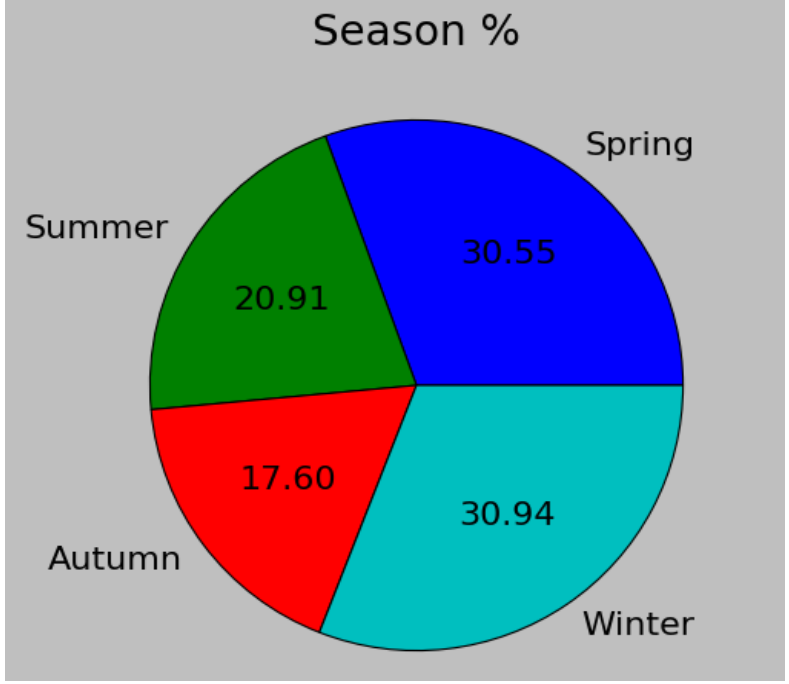
	AMB_TEMP	RAINFALL	RH	WS_HR	Season	DayPeriod	Target	WS_Change	WD_Change	PM10_25	PM25	WD_HR_gp
9	13.0	0.4	85.0	2.0	Q4	Morning	0.0	-0.1	14.0	17.0	35.0	EN
10	13.0	0.2	90.0	1.8	Q4	Morning	0.0	0.1	9.0	9.0	37.0	EN
11	13.0	1.4	90.0	2.4	Q4	Morning	0.0	0.4	2.0	6.0	34.0	EN

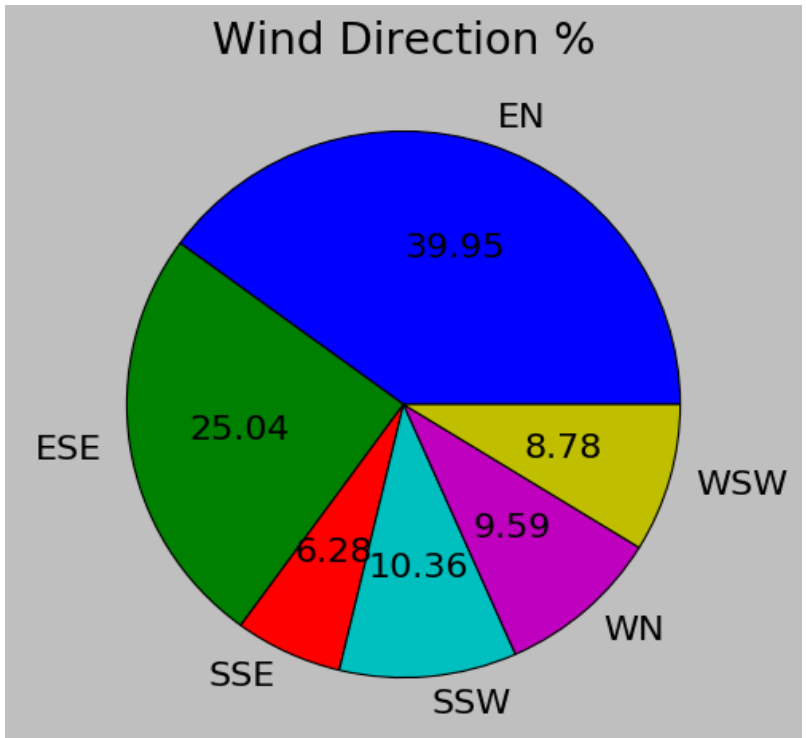
Plot categorical varibles distribution

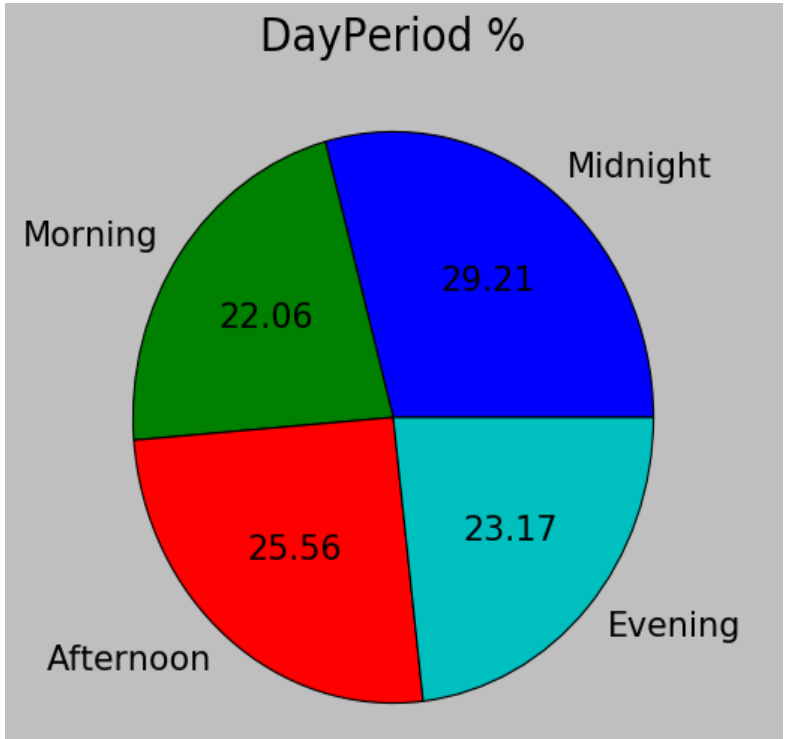
In [30]:

```
print("★☆ Plot Target with pie chart")
NR =TT[(TT['Target']==1)]
R = TT[(TT['Target']==0)]

series = pd.Series([NR.shape[0],R.shape[0]], index=['No Rain','Rain'],name='')
series.plot.pie(figsize=(7, 7), autopct='%.2f', fontsize=20)
pl.title("Target %", fontsize = 25)
pl.show()

print("★☆ Plot Season with pie chart")
Q1 =TT[(TT['Season']=='Q1')]
Q2 = TT[(TT['Season']=='Q2')]
Q3 =TT[(TT['Season']=='Q3')]
Q4 = TT[(TT['Season']=='Q4')]

series = pd.Series([Q1.shape[0],Q2.shape[0],Q3.shape[0],Q4.shape[0]], index=['Spring', 'Summer','Autumn','Winter'], name='')
series.plot.pie(figsize=(7, 7), autopct='%.2f', fontsize=20)
pl.title("Season %", fontsize = 25)
pl.show()

print("★☆ Plot DayPeriod with pie chart")
P1 =TT[(TT['DayPeriod']=='Midnight')]
P2 = TT[(TT['DayPeriod']=='Morning')]
P3 =TT[(TT['DayPeriod']=='Afternoon')]
P4 = TT[(TT['DayPeriod']=='Evening')]

series = pd.Series([P1.shape[0],P2.shape[0],P3.shape[0],P4.shape[0]], index=['Midnight', 'Morning','Afternoon','Evening'], name='
series.plot.pie(figsize=(7, 7), autopct='%.2f', fontsize=20)
pl.title("DayPeriod %", fontsize = 25)
pl.show()

print("★☆ Plot WIND_DIREC_gp with pie chart")
Gp =[]
for i in range(0,6):
    G =TT.groupby(['WD_HR_gp']).size()[i]
    Gp.append(G)
#TT.groupby(['WD_HR_gp']).size()

series = pd.Series(Gp, index=['EN','ESE','SSE','SSW','WN','WSW'], name='')
series.plot.pie(figsize=(7, 7), autopct='%.2f', fontsize=20)
pl.title("Wind Direction %", fontsize = 25)

pl.show()
```


Target %
No Rain
23.55
76.45
Rain


Season %
Spring
Summer
30.55
20.91
17.60
30.94
Autumn
Winter


Wind Direction %
EN
39.95
ESE
25.04
8.78
WSW
6.28
10.36
9.59
WN
SSE
SSW


DayPeriod %
Midnight
Morning
29.21
22.06
25.56
23.17
Afternoon
Evening

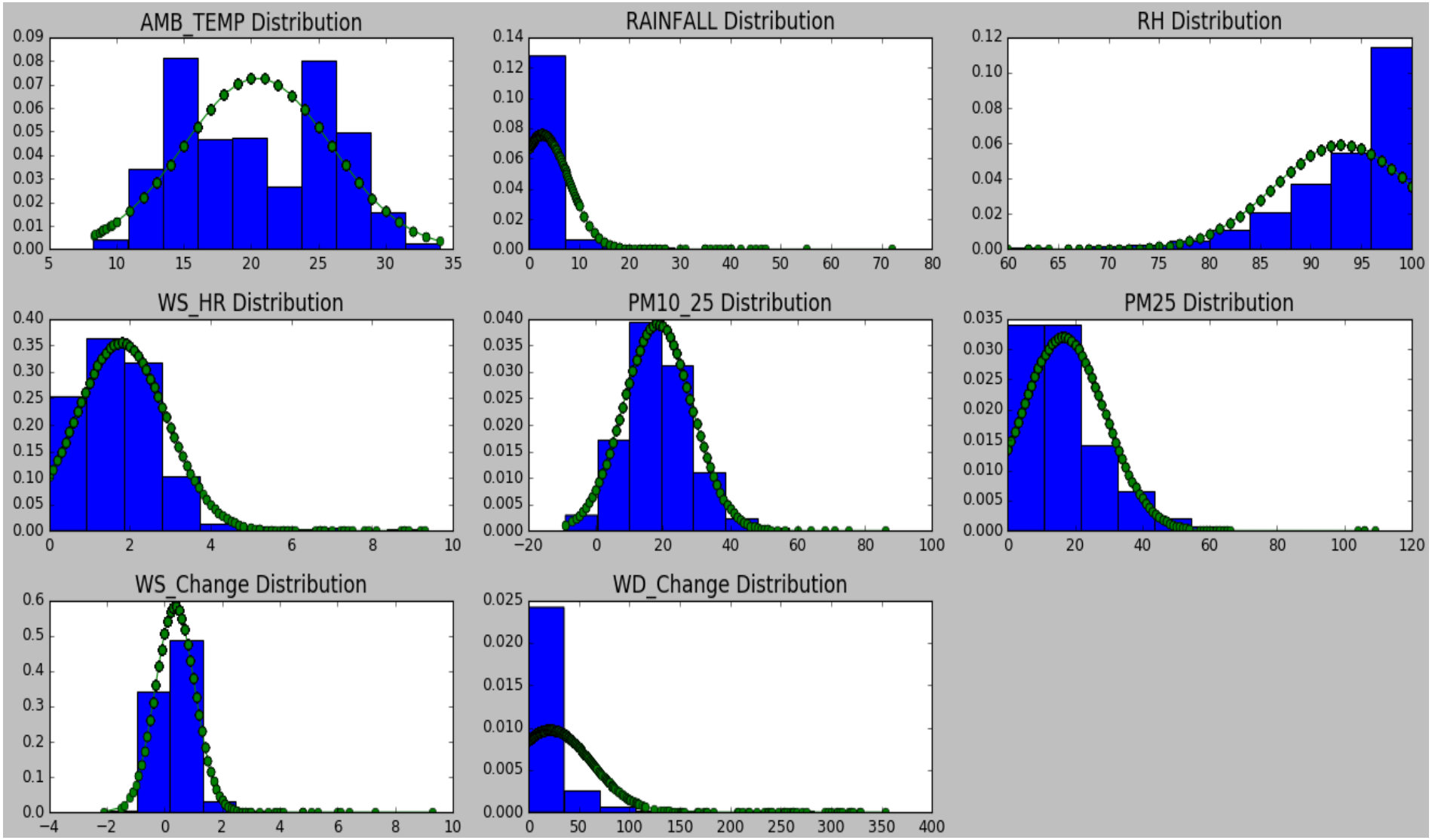
Plot numeric variables distribution

In [31]:
```
print("★☆ Plot Numeric Variable Distribution")
fig = pl.figure(figsize=(16.7, 8))
new_index = ['AMB_TEMP','RAINFALL','RH','WS_HR','PM10_25','PM25','Hour','Season','Target','WS_Change','WD_Change','WD_HR_gp']
num_index = ['AMB_TEMP','RAINFALL','RH','WS_HR','PM10_25','PM25','WS_Change','WD_Change']
cat_index = ['Hour','Season','Target','WD_HR_gp']
n=331
for i in num_index:
    h = sorted(list(TT[i]))
    fit = stats.norm.pdf(h, np.mean(h), np.std(h))  #this is a fitting indeed
    sub = fig.add_subplot(n)
    sub.hist(h,normed=True)
    sub.plot(h,fit,'-o')
    sub.set_title(i+' Distribution', fontsize=16)
    n = n+1

pl.tight_layout()
pl.show()

grouped = TT.groupby('Target')
grouped.agg([np.mean,np.std])
```

```
★☆ Plot Numeric Variable Distribution
```

Out[31]:

	AMB_TEMP		RAINFALL		RH		WS_HR		WS_Change		WD_Change		PM10_25
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean
Target													
0.0	20.193099	5.506194	2.890715	5.450141	93.552070	6.439646	1.816562	1.175888	0.382936	0.723490	21.800690	42.913582	17.921581
1.0	21.431568	5.312008	2.147047	4.490658	92.010183	7.512024	1.634623	0.924620	0.315071	0.534233	19.955601	33.542276	19.604888


AMB_TEMP Distribution
RAINFALL Distribution
RH Distribution
WS_HR Distribution
PM10_25 Distribution
PM25 Distribution
WS_Change Distribution
WD_Change Distribution

Detect rows meet outlier criteria

In [32]:
```
deO_TT = TT
deO_TT = deO_TT[((deO_TT.AMB_TEMP - deO_TT.AMB_TEMP.mean()) / deO_TT.AMB_TEMP.std()).abs() < 3]
deO_TT = deO_TT[((deO_TT.RAINFALL - deO_TT.RAINFALL.mean()) / deO_TT.RAINFALL.std()).abs() < 3]
deO_TT = deO_TT[((deO_TT.RH - deO_TT.RH.mean()) / deO_TT.RH.std()).abs() < 3]
deO_TT = deO_TT[((deO_TT.WS_HR - deO_TT.WS_HR.mean()) / deO_TT.WS_HR.std()).abs() < 3]
deO_TT = deO_TT[((deO_TT.PM10_25 - deO_TT.PM10_25.mean()) / deO_TT.PM10_25.std()).abs() < 3]
deO_TT = deO_TT[((deO_TT.PM25 - deO_TT.PM25.mean()) / deO_TT.PM25.std()).abs() < 3]
deO_TT = deO_TT[((deO_TT.WS_Change - deO_TT.WS_Change.mean()) / deO_TT.WS_Change.std()).abs() < 3]
deO_TT = deO_TT[((deO_TT.WD_Change - deO_TT.WD_Change.mean()) / deO_TT.WD_Change.std()).abs() < 3]
print("There are ", TT.shape[0]-deO_TT.shape[0], "(%.3f %%) rows meet the critirias of outlier. " %((TT.shape[0]-deO_TT.shape[0]

grouped = deO_TT.groupby('Target')
grouped.agg([np.mean,np.std])
```

```
There are  211 (10.120 %) rows meet the critirias of outlier.
```

Out[32]:

	AMB_TEMP		RAINFALL		RH		WS_HR		WS_Change		WD_Change		PM10_25
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean
Target													
0.0	19.654164	5.325758	2.127222	2.854741	94.216935	5.530204	1.724563	0.882481	0.299090	0.457209	15.767110	20.086841	17.370189
1.0	21.060449	5.066531	1.702921	2.625192	93.047191	6.103833	1.617303	0.845759	0.286966	0.471351	16.255955	19.552704	18.923596

Model split (90%, 10%)

In [33]:
```
features = ['AMB_TEMP','RAINFALL','RH','WS_HR','PM10_25','PM25','Season','DayPeriod','WS_Change','WD_Change','WD_HR_gp']
TT.index = range(len(TT))

X = TT[features]
y = TT['Target']

# A random permutation, to split the data randomly
np.random.seed(1234)
indices = np.random.permutation(len(TT))

# Split train and test data
idx1, idx2 = train_test_split(indices, test_size=0.1, random_state=0)
Train_data = TT.iloc[idx1]
Test_data = TT.iloc[idx2]
print("★☆ Train_data : rows : ", Train_data.shape[0], "; columns : ", Train_data.shape[1])
print(Train_data.groupby(['Target']).size())
print()
print("★☆ Test_data : rows : ", Test_data.shape[0], "; columns : ", Test_data.shape[1])
print(Test_data.groupby(['Target']).size())
```

```
★☆ Train_data : rows :  1876 ; columns :  12
Target
0.0    1439
1.0     437
dtype: int64

★☆ Test_data : rows :  209 ; columns :  12
Target
0.0    155
1.0     54
dtype: int64
```

In [34]:
```
y=0
X=0
def LogisticReg_Xy(Train_data, Test_data):
    yt, Xt = dmatrices('Target ~ AMB_TEMP + RAINFALL + RH + WS_HR + PM10_25 + PM25 + WS_Change + WD_Change + C(Season) + C(DayPer
                       Train_data, return_type="dataframe")

    Xt = Xt.rename(columns = {'C(Season)[T.Q2]':'Season_Q2',
                              'C(Season)[T.Q3]':'Season_Q3',
                              'C(Season)[T.Q4]':'Season_Q4',
                              'C(DayPeriod)[T.Evening]':'DayP_Evening',
                              'C(DayPeriod)[T.Midnight]':'DayP_Midnight',
                              'C(DayPeriod)[T.Morning]':'DayP_Morning',
                              'C(WD_HR_gp)[T.ESE]':'WDHRgp_ESE',
                              'C(WD_HR_gp)[T.SSE]':'WDHRgp_SSE',
                              'C(WD_HR_gp)[T.SSW]':'WDHRgp_SSW',
                              'C(WD_HR_gp)[T.WN]':'WDHRgp_WN',
                              'C(WD_HR_gp)[T.WSW]':'WDHRgp_WSW'})

    yt = np.ravel(yt)

    yw, Xw = dmatrices('Target ~ AMB_TEMP + RAINFALL + RH + WS_HR + PM10_25 + PM25 + WS_Change + WD_Change + C(Season) + C(DayPer
                       Test_data, return_type="dataframe")

    Xw = Xw.rename(columns = {'C(Season)[T.Q2]':'Season_Q2',
                              'C(Season)[T.Q3]':'Season_Q3',
                              'C(Season)[T.Q4]':'Season_Q4',
                              'C(DayPeriod)[T.Evening]':'DayP_Evening',
                              'C(DayPeriod)[T.Midnight]':'DayP_Midnight',
                              'C(DayPeriod)[T.Morning]':'DayP_Morning',
                              'C(WD_HR_gp)[T.ESE]':'WDHRgp_ESE',
                              'C(WD_HR_gp)[T.SSE]':'WDHRgp_SSE',
                              'C(WD_HR_gp)[T.SSW]':'WDHRgp_SSW',
                              'C(WD_HR_gp)[T.WN]':'WDHRgp_WN',
                              'C(WD_HR_gp)[T.WSW]':'WDHRgp_WSW'})

    yw = np.ravel(yw)
    return yt, Xt, yw, Xw

# Trainin/Testing data X y create
yt, Xt, yw, Xw = LogisticReg_Xy(Train_data, Test_data)
Xt[:5]
```

Logistic regression

Training data performance

In [37]:
```
model = LogisticRegression()
model = model.fit(Xt, yt)
print("★☆ Training model accuracy: %.2f " % model.score(Xt, yt))

# evaluate the model using 10-fold cross-validation
scores = cross_val_score(LogisticRegression(), Xt, yt, scoring='accuracy', cv=10)
print ("★☆ 10-fold cross-validation accuracies: \n", scores)
```

```
★☆ Training model accuracy: 0.77
★☆ 10-fold cross-validation accuracies:
 [ 0.7606383   0.7712766   0.76595745  0.76595745  0.7606383   0.7712766
  0.7712766   0.77540107  0.78074866  0.7688172 ]
```

Testing data performance

In [38]:
```
predicted = model.predict(Xw)
print("★☆ Test model accuracy: %.2f " % metrics.roc_auc_score(yw, predicted))
prob = model.predict_proba(Xw)[:, 1]
Section10Performance(prob, yw)
```

```
★☆ Test model accuracy: 0.50
```

Decision tree

In [39]:
```
from sklearn.cross_validation import KFold
from sklearn import tree
```

Training data performance

In [40]:
```
dt = tree.DecisionTreeClassifier(max_depth=8, random_state=0)
dt = dt.fit(Xt, yt)
print("★☆ Training model accuracy: %.2f " % dt.score(Xt, yt))
print()
# evaluate the model using 10-fold cross-validation
scores = cross_val_score(tree.DecisionTreeClassifier(max_depth=8,random_state=0), Xt, yt, scoring='accuracy', cv=10)
print ("★☆ 10-fold cross-validation accuracies: \n", scores)
```

```
★☆ Training model accuracy: 0.82

★☆ 10-fold cross-validation accuracies:
 [ 0.7287234   0.7606383   0.75531915  0.71276596  0.74468085  0.71276596
  0.70744681  0.68983957  0.76470588  0.69892473]
```

Testing data performance

In [41]:
```
predict = dt.predict(Xw)
print("★☆ Test model accuracy: %.2f " % metrics.roc_auc_score(yw, predict))
print()
prob = dt.predict_proba(Xw)[:, 1]
Section10Performance(prob, yw)
```

```
★☆ Test model accuracy: 0.54
```