CODING DOJO ACADEMY

الأكاديمية السعودية الرقمية
SAUDI DIGITAL ACADEMY

# Weather in Saudi Arabia

Data Miners Group
Raheed Albaloud (Leader)
Razan Albishri
Bashayer Alahmari
Suha Shafi
Ibrahim Alsufyani

# TABLE OF CONTENTS


·01·
Dataset choice
What is the Domain of our Dataset?

·02·
Cleaning
Reshaping And Cleaning The Dataset

·03·
Performing EDA
Plots

·04·
Machine Learning
Splitting, feature, models

·05·
Hyperparameter Tuning
Proper hyperparameters tuning

·06·
Pipelines
Implementation of pipelines

# Saudi Arabia Weather Dataset

**weather-sa-2017-2019-clean.csv** (18.81 MB)

Detail | Compact | Column

10 of 15 columns

| city | date | time | year | month | day | hour | minute | weather |
|---|---|---|---|---|---|---|---|---|
| Qassim | 1 January 2017 | 00:00 | 2017 | 1 | 1 | 24 | 0 | Clear |
| Qassim | 1 January 2017 | 01:00 | 2017 | 1 | 1 | 1 | 0 | Clear |
| Qassim | 1 January 2017 | 03:00 | 2017 | 1 | 1 | 3 | 0 | Clear |
| Qassim | 1 January 2017 | 04:00 | 2017 | 1 | 1 | 4 | 0 | Clear |
| Qassim | 1 January 2017 | 05:00 | 2017 | 1 | 1 | 5 | 0 | Clear |
| Qassim | 1 January 2017 | 06:00 | 2017 | 1 | 1 | 6 | 0 | Clear |
| Qassim | 1 January 2017 | 07:00 | 2017 | 1 | 1 | 7 | 0 | Sunny |
| Qassim | 1 January 2017 | 08:00 | 2017 | 1 | 1 | 8 | 0 | Sunny |
| Qassim | 1 January 2017 | 09:00 | 2017 | 1 | 1 | 9 | 0 | Sunny |
| Qassim | 1 January 2017 | 10:00 | 2017 | 1 | 1 | 10 | 0 | Sunny |
| Qassim | 1 January 2017 | 11:00 | 2017 | 1 | 1 | 11 | 0 | Sunny |
| Qassim | 1 January 2017 | 12:00 | 2017 | 1 | 1 | 12 | 0 | Sunny |
| Qassim | 1 January 2017 | 13:00 | 2017 | 1 | 1 | 13 | 0 | Sunny |
| Qassim | 1 January 2017 | 14:00 | 2017 | 1 | 1 | 14 | 0 | Sunny |
| Qassim | 1 January 2017 | 15:00 | 2017 | 1 | 1 | 15 | 0 | Sunny |
| Qassim | 1 January 2017 | 16:00 | 2017 | 1 | 1 | 16 | 0 | Sunny |
| Qassim | 1 January 2017 | 17:00 | 2017 | 1 | 1 | 17 | 0 | Sunny |
| Qassim | 1 January 2017 | 18:00 | 2017 | 1 | 1 | 18 | 0 | Clear |

# Load the dataset

In [2]:
```
weatherKSA = pd.read_csv('D:\Downloads\weather-sa-2017-2019-clean.csv')
weatherKSA.head()
```

Out[2]:

| | Unnamed: 0 | city | date | time | year | month | day | hour | minute | weather | temp | wind | humidity | barometer | visibility |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | Qassim | 1 January 2017 | 00:00 | 2017 | 1 | 1 | 24 | 0 | Clear | 17 | 11 | 64% | 1018.0 | 16 |
| **1** | 1 | Qassim | 1 January 2017 | 01:00 | 2017 | 1 | 1 | 1 | 0 | Clear | 17 | 6 | 64% | 1018.0 | 16 |
| **2** | 2 | Qassim | 1 January 2017 | 03:00 | 2017 | 1 | 1 | 3 | 0 | Clear | 15 | 11 | 72% | 1019.0 | 16 |
| **3** | 3 | Qassim | 1 January 2017 | 04:00 | 2017 | 1 | 1 | 4 | 0 | Clear | 15 | 11 | 72% | 1019.0 | 16 |
| **4** | 4 | Qassim | 1 January 2017 | 05:00 | 2017 | 1 | 1 | 5 | 0 | Clear | 15 | 9 | 72% | 1019.0 | 16 |

# Reshape and Clean The Dataset

# Shape of the Dataset

```
In [3]: print('Shape of the data is: ',weatherKSA.shape)

        Shape of the data is:  (249023, 15)
```

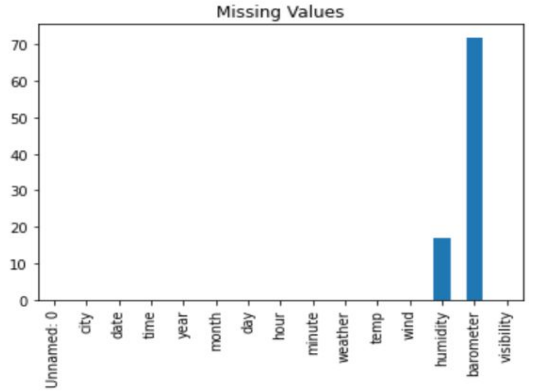
# Check if there are missing values

```
In [4]: weatherKSA.isnull().sum()
```

```
Out[4]: Unnamed: 0     0
        city           0
        date           0
        time           0
        year           0
        month          0
        day            0
        hour           0
        minute         0
        weather        0
        temp           0
        wind           0
        humidity      17
        barometer     72
        visibility     0
        dtype: int64
```

```
In [5]: weatherKSA.isnull().sum().plot(kind="bar", title="Missing Values")
```

```
Out[5]: <AxesSubplot:title={'center':'Missing Values'}>
```



# Drop and Replacement the NA values

```
In [6]: weatherKSA.drop('Unnamed: 0', axis=1, inplace = True)

In [8]: weatherKSA['humidity'].fillna(method='ffill',inplace=True)

In [9]: weatherKSA['barometer'].fillna(method='ffill',inplace=True)

In [7]: weatherKSA.columns

Out[7]: Index(['city', 'date', 'time', 'year', 'month', 'day', 'hour', 'minute',
               'weather', 'temp', 'wind', 'humidity', 'barometer', 'visibility'],
              dtype='object')
```

# Check if there are missing values

```
In [12]: weatherKSA.isnull().sum()

Out[12]: city          0
         date          0
         time          0
         year          0
         month         0
         day           0
         hour          0
         minute        0
         weather       0
         temp          0
         wind          0
         humidity      0
         barometer     0
         visibility    0
         dtype: int64
```

```
In [11]: weatherKSA.isnull().sum().plot(kind="bar", title="Missing Values")

Out[11]: <AxesSubplot:title={'center':'Missing Values'}>
```

# Description of Dataset

```
In [13]: weatherKSA.describe().T
```

Out[13]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **year** | 249023.0 | 2017.710007 | 0.706113 | 2017.0 | 2017.0 | 2018.0 | 2018.0 | 2019.0 |
| **month** | 249023.0 | 6.050694 | 3.521591 | 1.0 | 3.0 | 6.0 | 9.0 | 12.0 |
| **day** | 249023.0 | 15.691081 | 8.787958 | 1.0 | 8.0 | 16.0 | 23.0 | 31.0 |
| **hour** | 249023.0 | 12.536890 | 6.910254 | 1.0 | 7.0 | 13.0 | 19.0 | 24.0 |
| **minute** | 249023.0 | 0.131108 | 1.970710 | 0.0 | 0.0 | 0.0 | 0.0 | 59.0 |
| **temp** | 249023.0 | 24.722624 | 8.880913 | -4.0 | 18.0 | 24.0 | 31.0 | 50.0 |
| **wind** | 249023.0 | 12.957104 | 8.711619 | -1.0 | 7.0 | 11.0 | 19.0 | 163.0 |
| **barometer** | 249023.0 | 1015.452557 | 6.973451 | 904.0 | 1011.0 | 1016.0 | 1021.0 | 1101.0 |
| **visibility** | 249023.0 | 11.053453 | 7.053005 | -1.0 | 5.0 | 16.0 | 16.0 | 161.0 |

# Description of Dataset

```
In [14]: weatherKSA.describe(exclude = 'number').T
```

Out[14]:

| | count | unique | top | freq |
|---|---|---|---|---|
| **city** | 249023 | 13 | Jawf | 20352 |
| **date** | 249023 | 850 | 24 November 2018 | 321 |
| **time** | 249023 | 710 | 07:00 | 10415 |
| **weather** | 249023 | 81 | Clear | 98827 |
| **humidity** | 249023 | 92 | 13% | 6903 |

# Dataset Information

```
In [15]: weatherKSA.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 249023 entries, 0 to 249022
Data columns (total 14 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   city        249023 non-null  object
 1   date        249023 non-null  object
 2   time        249023 non-null  object
 3   year        249023 non-null  int64
 4   month       249023 non-null  int64
 5   day         249023 non-null  int64
 6   hour        249023 non-null  int64
 7   minute      249023 non-null  int64
 8   weather     249023 non-null  object
 9   temp        249023 non-null  int64
 10  wind        249023 non-null  int64
 11  humidity    249023 non-null  object
 12  barometer   249023 non-null  float64
 13  visibility  249023 non-null  int64
dtypes: float64(1), int64(8), object(5)
memory usage: 26.6+ MB
```

```
In [17]: weatherKSA["date"] = pd.to_datetime(weatherKSA["date"])
```

```
In [19]: weatherKSA.info()
```

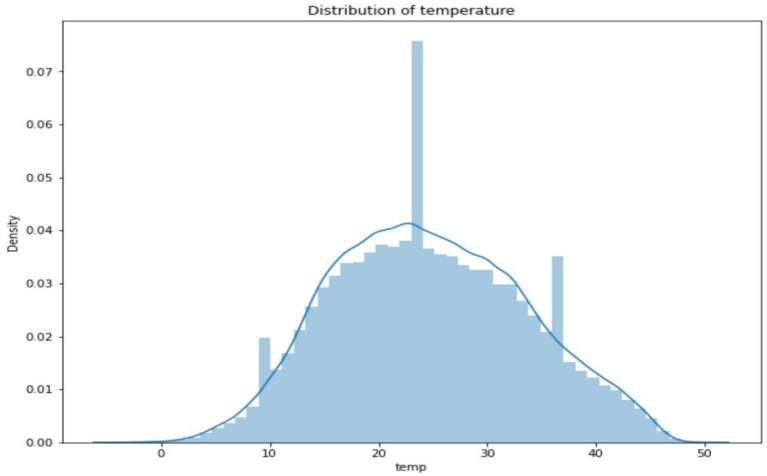
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 249023 entries, 0 to 249022
Data columns (total 14 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   city        249023 non-null  object
 1   date        249023 non-null  datetime64[ns]
 2   time        249023 non-null  object
 3   year        249023 non-null  int64
 4   month       249023 non-null  int64
 5   day         249023 non-null  int64
 6   hour        249023 non-null  int64
 7   minute      249023 non-null  int64
 8   weather     249023 non-null  object
 9   temp        249023 non-null  int64
 10  wind        249023 non-null  int64
 11  humidity    249023 non-null  object
 12  barometer   249023 non-null  float64
 13  visibility  249023 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(8), object(4)
memory usage: 26.6+ MB
```

# Performing EDA

# 1st Plot

In [20]:

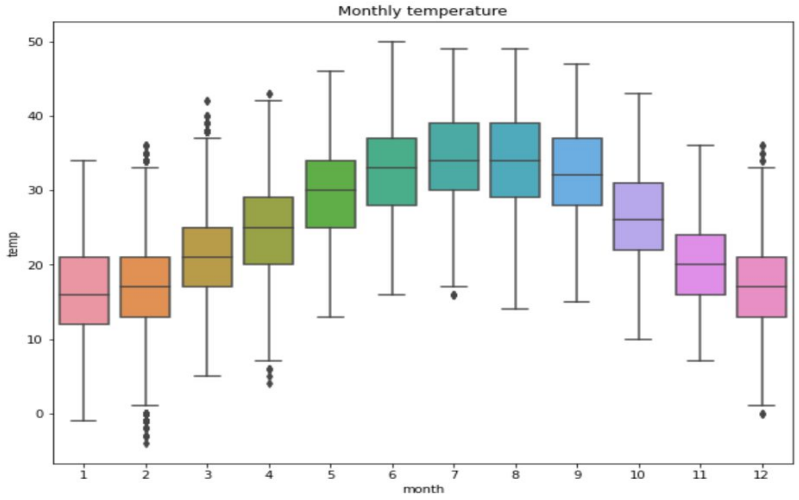
```
plt.figure(figsize=(10,8))
plt.title("Distribution of temperature")
sns.distplot(weatherKSA.temp);
```



# 2nd Plot

In [21]:

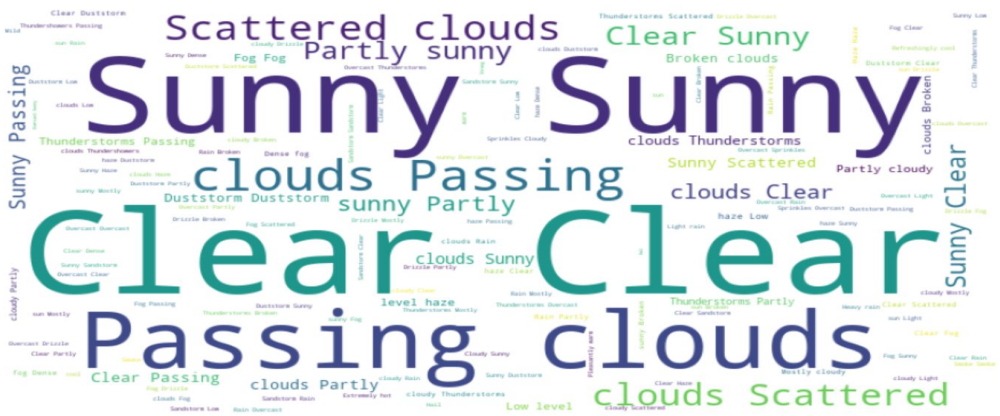
```
plt.figure(figsize=(10,8))
plt.title("Monthly temperature")
sns.boxplot(weatherKSA.month, weatherKSA.temp);
```



# 3rd Plot

In [22]:

```
words=' '.join((event for event in weatherKSA.weather.dropna() ))
wc=WordCloud(width=800,height=400,background_color='white').generate(words)
plt.figure(figsize=(16,8))
plt.axis("off")
plt.grid(False)
plt.imshow(wc);
```



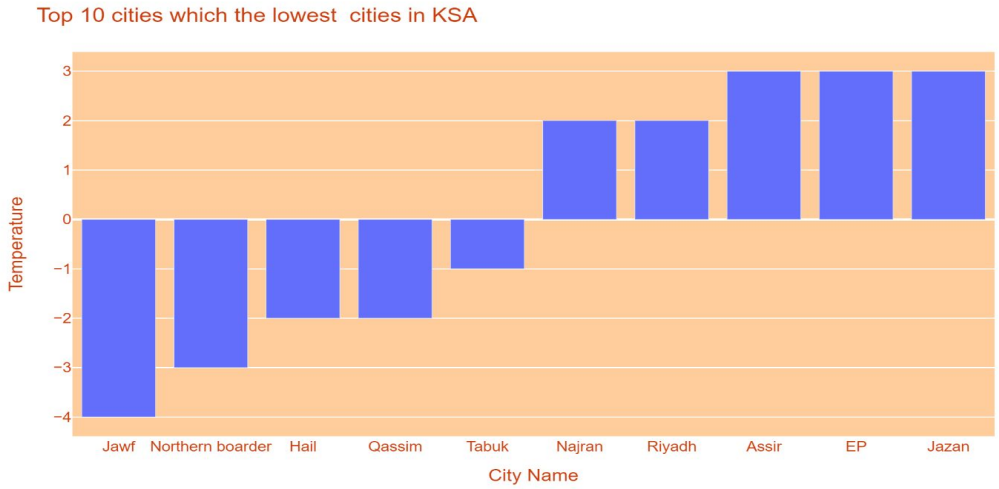
# 4th Plot

In [23]:
```
min_temp=weatherKSA.groupby('city')['temp'].min().sort_values()
min_temp.head(10).index
```

Out[23]:
```
Index(['Jawf', 'Northern boarder', 'Hail', 'Qassim', 'Tabuk', 'Najran',
       'Riyadh', 'Assir', 'EP', 'Jazan'],
      dtype='object', name='city')
```

In [24]:
```
fig=px.bar(y=min_temp.head(10),x=min_temp.head(10).index)
fig.update_layout(
    title="Top 10 cities which the lowest  cities in KSA",
    xaxis_title="City Name",
    yaxis_title="Temperature",
   #legend_title="",
    plot_bgcolor='#ffcc9c',
    font=dict(
        family="Arial",
        size=14,
        color="#cc3e0e"
    )
)
```

# 4th Plot


Top 10 cities which the lowest cities in KSA
Temperature
3
2
1
0
−1
−2
−3
−4
Jawf
Northern boarder
Hail
Qassim
Tabuk
Najran
Riyadh
Assir
EP
Jazan
City Name

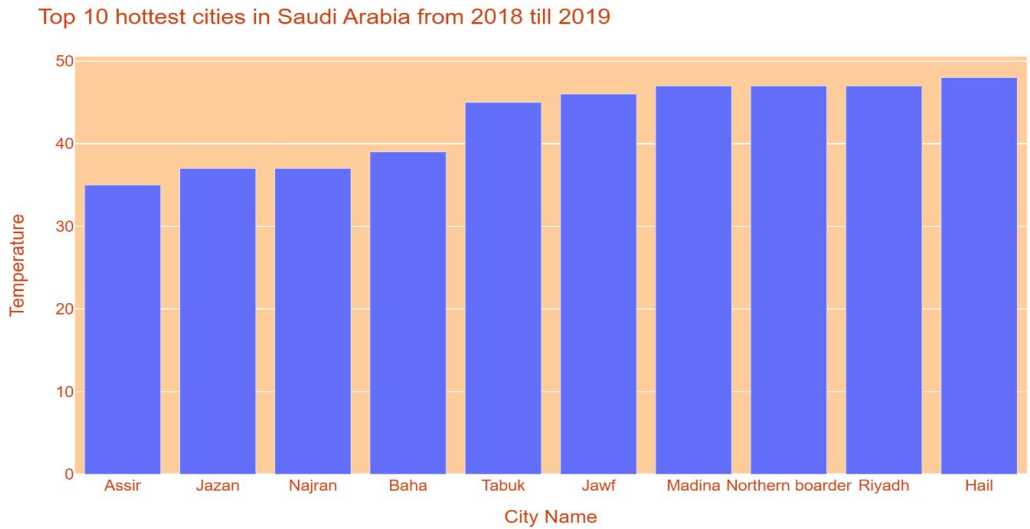
# 5th Plot

In [25]:
```
max_temp=weatherKSA.groupby('city')['temp'].max().sort_values()
max_temp.head(10).index
```

Out[25]:
```
Index(['Assir', 'Jazan', 'Najran', 'Baha', 'Tabuk', 'Jawf', 'Madina',
       'Northern boarder', 'Riyadh', 'Hail'],
      dtype='object', name='city')
```

In [26]:
```
fig=px.bar(y=max_temp.head(10),x=max_temp.head(10).index)
fig.update_layout(
    title="Top 10 hottest cities in Saudi Arabia from 2018 till 2019",
    xaxis_title="City Name",
    yaxis_title="Temperature",
    #legend_title="",
    plot_bgcolor='#ffcc9c',
    font=dict(
        family="Arial",
        size=14,
        color="#cc3e0e"
    )
)
```

# 5th Plot

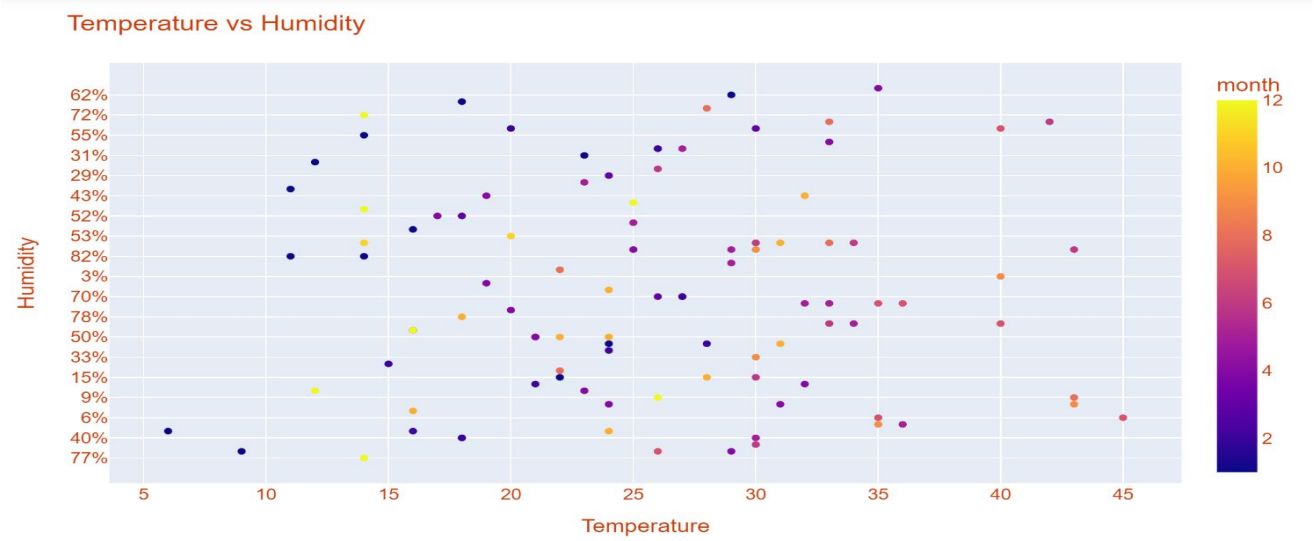

Top 10 hottest cities in Saudi Arabia from 2018 till 2019
Temperature
50
40
30
20
10
0
Assir
Jazan
Najran
Baha
Tabuk
Jawf
Madina
Northern boarder
Riyadh
Hail
City Name

# 6th Plot

In [27]:
```
sample_df=weatherKSA.sample(100)
```

In [28]:
```
fig=px.scatter(sample_df, x='temp',y='humidity',color='month',hover_name='city')
fig.update_layout(
    title="Temperature vs Humidity",
    xaxis_title="Temperature",
    yaxis_title="Humidity",
    legend_title="Month",
    font=dict(
        family="Arial",
        size=14,
        color="#cc3e0e"
    )
)
fig
```

# 6th Plot


Temperature vs Humidity
Humidity
62%
72%
55%
31%
29%
43%
52%
53%
82%
3%
70%
78%
50%
33%
15%
9%
6%
40%
77%
5
10
15
20
25
30
35
40
45
Temperature
month
12
10
8
6
4
2

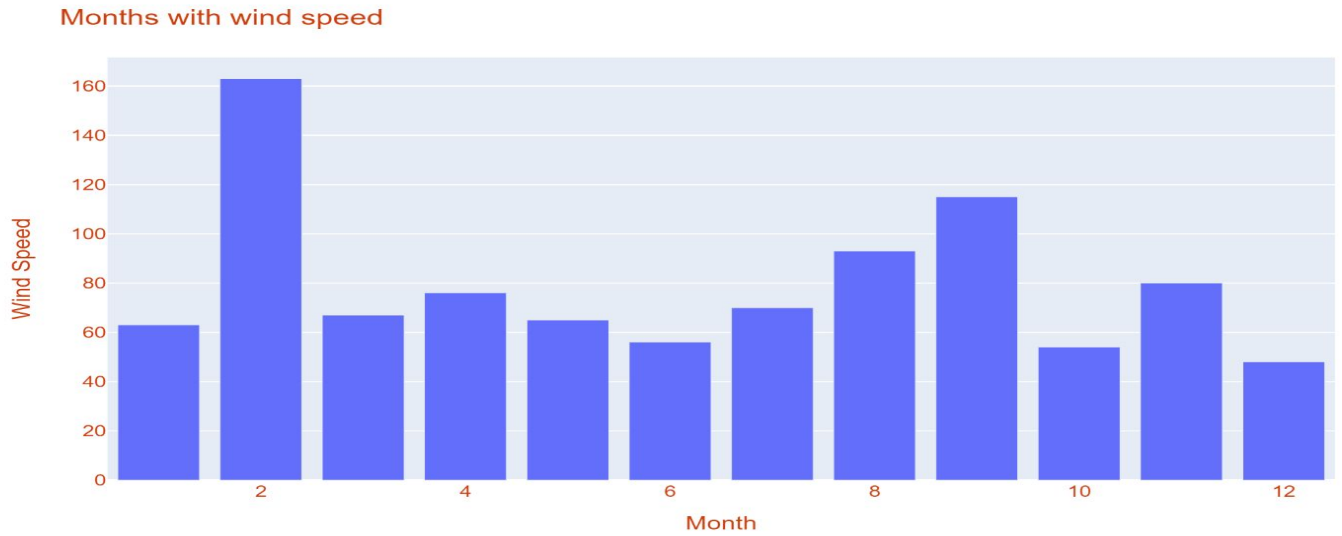
# 7th Plot

In [29]:
```
max_wind=weatherKSA.groupby('month')['wind'].max().sort_values(ascending=False)
max_wind.index
```

Out[29]:
```
Int64Index([2, 9, 8, 11, 4, 7, 3, 5, 1, 6, 10, 12], dtype='int64', name='month')
```

In [30]:
```
#plot the months with wind speed
fig=px.bar(y=max_wind,x=max_wind.index)
fig.update_layout(
    title="Months with wind speed",
    xaxis_title="Month",
    yaxis_title="Wind Speed",
        font=dict(
        family="Arial",
        size=14,
        color="#cc3e0e"
    )
)
fig
```

# 7th Plot


Months with wind speed
Wind Speed
160
140
120
100
80
60
40
20
0
2
4
6
8
10
12
Month

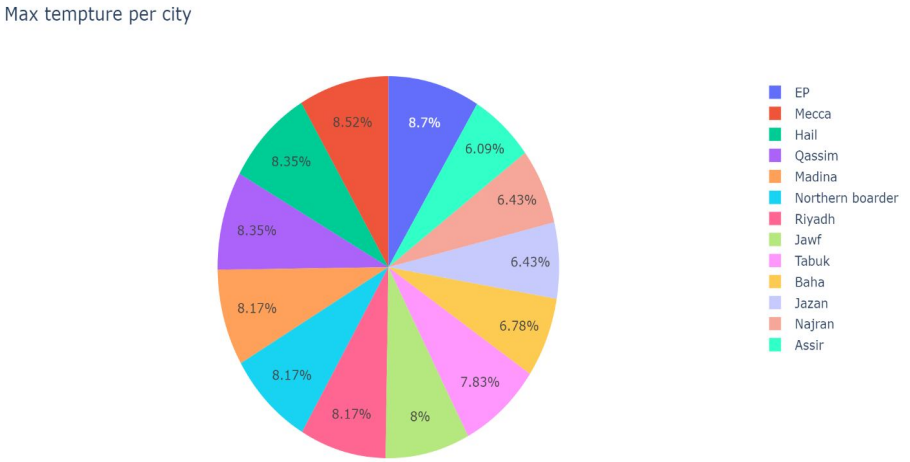
# 8th Plot

```
In [32]: df = weatherKSA.groupby('city')[['temp']].max()
         df.reset_index(inplace=True)
         df
```

Out[32]:

| | city | temp |
|---|---|---|
| 0 | Assir | 35 |
| 1 | Baha | 39 |
| 2 | EP | 50 |
| 3 | Hail | 48 |
| 4 | Jawf | 46 |
| 5 | Jazan | 37 |
| 6 | Madina | 47 |
| 7 | Mecca | 49 |
| 8 | Najran | 37 |
| 9 | Northern boarder | 47 |
| 10 | Qassim | 48 |
| 11 | Riyadh | 47 |
| 12 | Tabuk | 45 |

```
In [33]: fig = px.pie(df, values='temp', names= 'city', title='Max tempture per city')
         fig.show()
```



# 9th Plot

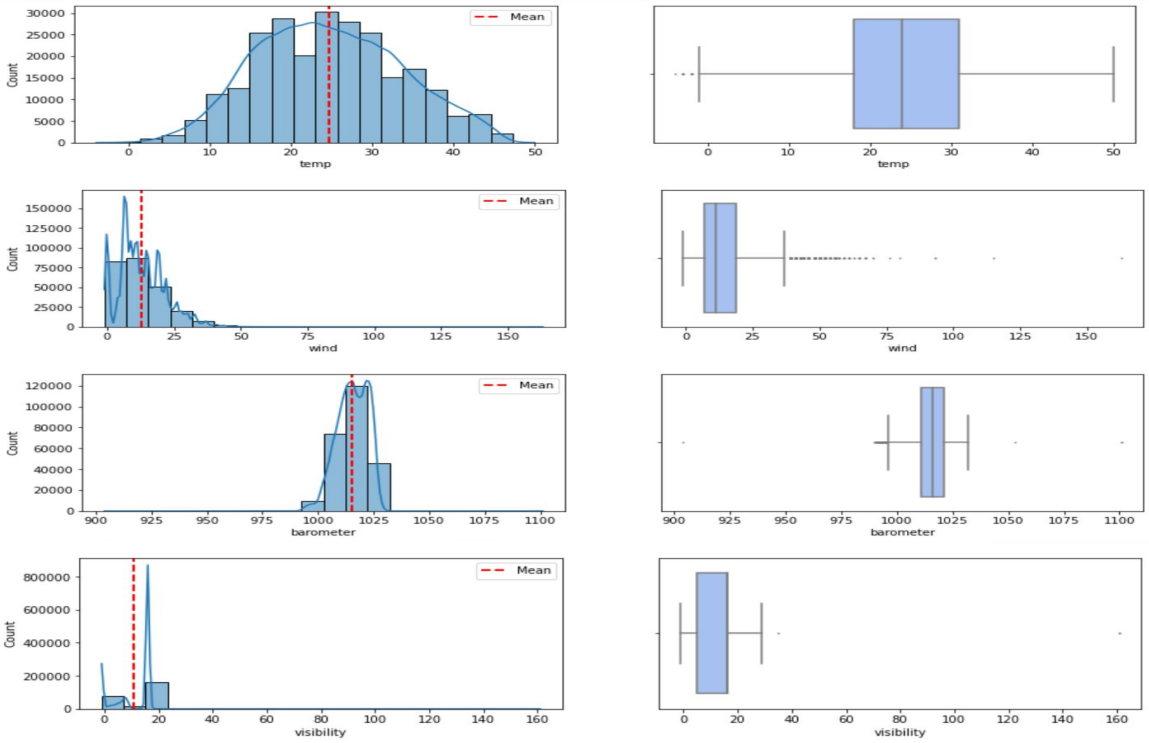
In [31]:

```
num_cols=weatherKSA.loc[:,'temp':'visibility'].select_dtypes(include=['int64','float64']).columns.tolist()
for i in num_cols:

    fig, axs = plt.subplots(1,2,figsize=(15, 3))

    sns.histplot(weatherKSA[i],bins=20, kde=True,ax=axs[0]);
    sns.boxplot(x= weatherKSA[i], ax = axs[1], color='#99befd', fliersize=1);

    axs[0].axvline(weatherKSA[i].mean(), color='r', linewidth=2, linestyle='--', label='Mean')
    axs[0].legend()
```


Mean
Count
temp
wind
barometer
visibility

# Machine Learning

**weather-sa-2017-2019-clean.csv** (18.81 MB)

Detail | Compact | Column

10 of 15 columns

| city | date | time | year | month | day | hour | minute | weather |
|---|---|---|---|---|---|---|---|---|
| Qassim | 1 January 2017 | 00:00 | 2017 | 1 | 1 | 24 | 0 | Clear |
| Qassim | 1 January 2017 | 01:00 | 2017 | 1 | 1 | 1 | 0 | Clear |
| Qassim | 1 January 2017 | 03:00 | 2017 | 1 | 1 | 3 | 0 | Clear |
| Qassim | 1 January 2017 | 04:00 | 2017 | 1 | 1 | 4 | 0 | Clear |
| Qassim | 1 January 2017 | 05:00 | 2017 | 1 | 1 | 5 | 0 | Clear |
| Qassim | 1 January 2017 | 06:00 | 2017 | 1 | 1 | 6 | 0 | Clear |
| Qassim | 1 January 2017 | 07:00 | 2017 | 1 | 1 | 7 | 0 | Sunny |
| Qassim | 1 January 2017 | 08:00 | 2017 | 1 | 1 | 8 | 0 | Sunny |
| Qassim | 1 January 2017 | 09:00 | 2017 | 1 | 1 | 9 | 0 | Sunny |
| Qassim | 1 January 2017 | 10:00 | 2017 | 1 | 1 | 10 | 0 | Sunny |
| Qassim | 1 January 2017 | 11:00 | 2017 | 1 | 1 | 11 | 0 | Sunny |
| Qassim | 1 January 2017 | 12:00 | 2017 | 1 | 1 | 12 | 0 | Sunny |
| Qassim | 1 January 2017 | 13:00 | 2017 | 1 | 1 | 13 | 0 | Sunny |
| Qassim | 1 January 2017 | 14:00 | 2017 | 1 | 1 | 14 | 0 | Sunny |
| Qassim | 1 January 2017 | 15:00 | 2017 | 1 | 1 | 15 | 0 | Sunny |
| Qassim | 1 January 2017 | 16:00 | 2017 | 1 | 1 | 16 | 0 | Sunny |
| Qassim | 1 January 2017 | 17:00 | 2017 | 1 | 1 | 17 | 0 | Sunny |
| Qassim | 1 January 2017 | 18:00 | 2017 | 1 | 1 | 18 | 0 | Clear |

# Transform all string values to integers

In [34]:
```
weatherKSA['weather']=weatherKSA['weather'].astype('category')
weatherKSA['weather']=weatherKSA['weather'].cat.codes
```

In [35]:
```
weatherKSA['city']=weatherKSA['city'].astype('category')
weatherKSA['city']=weatherKSA['city'].cat.codes
```

In [36]:
```
weatherKSA
```

Out[36]:

| | city | date | time | year | month | day | hour | minute | weather | temp | wind | humidity | barometer | visibility |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 10 | 2017-01-01 | 00:00 | 2017 | 1 | 1 | 24 | 0 | 1 | 17 | 11 | 64% | 1018.0 | 16 |
| **1** | 10 | 2017-01-01 | 01:00 | 2017 | 1 | 1 | 1 | 0 | 1 | 17 | 6 | 64% | 1018.0 | 16 |
| **2** | 10 | 2017-01-01 | 03:00 | 2017 | 1 | 1 | 3 | 0 | 1 | 15 | 11 | 72% | 1019.0 | 16 |
| **3** | 10 | 2017-01-01 | 04:00 | 2017 | 1 | 1 | 4 | 0 | 1 | 15 | 11 | 72% | 1019.0 | 16 |
| **4** | 10 | 2017-01-01 | 05:00 | 2017 | 1 | 1 | 5 | 0 | 1 | 15 | 9 | 72% | 1019.0 | 16 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **249018** | 4 | 2019-04-30 | 19:00 | 2019 | 4 | 30 | 19 | 0 | 40 | 32 | 19 | 14% | 1014.0 | -1 |
| **249019** | 4 | 2019-04-30 | 20:00 | 2019 | 4 | 30 | 20 | 0 | 40 | 29 | 9 | 22% | 1015.0 | -1 |
| **249020** | 4 | 2019-04-30 | 21:00 | 2019 | 4 | 30 | 21 | 0 | 40 | 27 | 7 | 24% | 1016.0 | -1 |
| **249021** | 4 | 2019-04-30 | 22:00 | 2019 | 4 | 30 | 22 | 0 | 1 | 26 | 0 | 26% | 1017.0 | 16 |
| **249022** | 4 | 2019-04-30 | 23:00 | 2019 | 4 | 30 | 23 | 0 | 1 | 24 | 7 | 29% | 1017.0 | 16 |

249023 rows × 14 columns

# Baseline Model

```
In [37]: weatherKSA["weather"].value_counts(normalize=True)
```

```
Out[37]: 1     0.396859
         65    0.330066
         40    0.138060
         56    0.061456
         39    0.027809
                 ...
         19    0.000004
         63    0.000004
         15    0.000004
         62    0.000004
         43    0.000004
         Name: weather, Length: 81, dtype: float64
```

# Feature Selection

In [38]:
```
features = ['city','temp','wind','barometer','visibility']
x = weatherKSA.loc[:, features]
y = weatherKSA.loc[:, ['weather']]
```

In [39]:
```
x.shape
```

Out[39]: (249023, 5)

In [40]:
```
y.shape
```

Out[40]: (249023, 1)

In [41]:
```
le = LabelEncoder()

x.iloc[:,0] = le.fit_transform(x.iloc[:,0])
```

# Split the Data

In [42]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

In [43]:
```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

# Machine Learning

**1st model: DecisionTreeClassifier**

In [44]:
```
from sklearn.linear_model import LogisticRegression
```

In [45]:
```
classifier_log=LogisticRegression(random_state=0)
classifier_log.fit(x_train,y_train)
```

Out[45]:
```
LogisticRegression(random_state=0)
```

In [46]:
```
class_tree = DecisionTreeClassifier(criterion = 'gini', random_state = 45).fit(x_train, y_train)
preds_class = class_tree.predict(x_test)

val_train = round(class_tree.score(x_train, y_train),2)*100
val_test = round(class_tree.score(x_test, y_test),2)*100

print(f'Training Accuracy: {val_train}%')
print(f'Test Accuracy: {val_test}%')
```

```
Training Accuracy: 86.0%
Test Accuracy: 64.0%
```

# Machine Learning

**2nd model: Random Forest Classifier**

In [47]:

```
class_forest = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
class_forest.fit(x_train, y_train)
preds_class = class_forest.predict(x_test)

val_train = round(class_forest.score(x_train, y_train),2)*100
val_test = round(class_forest.score(x_test, y_test),2)*100

print(f'Training Accuracy: {val_train}%')
print(f'Test Accuracy: {val_test}%')
```

```
Training Accuracy: 85.0%
Test Accuracy: 65.0%
```

# Machine Learning

**3rd model: Decision Tree Classifier**

In [48]:

```
clf3 = DecisionTreeClassifier(criterion = 'gini', random_state = 0, max_depth=4)
clf3.fit(x_train, y_train)
preds_class = clf3.predict(x_test)

val_train = round(clf3.score(x_train, y_train),2)*100
val_test = round(clf3.score(x_test, y_test),2)*100

print(f'Training Accuracy: {val_train}%')
print(f'Test Accuracy: {val_test}%')
```

```
Training Accuracy: 59.0%
Test Accuracy: 59.0%
```

# Hyperparameter

In [49]:

```
n_estimators = [5,20,50,100]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 120, num = 12)]
min_samples_split = [2, 6, 10]
min_samples_leaf = [1, 3, 4]
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators,

'max_features': max_features,

'max_depth': max_depth,

'min_samples_split': min_samples_split,

'min_samples_leaf': min_samples_leaf,

'bootstrap': bootstrap}
```

# Hyperparameter

In [50]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state = 50)
```

In [51]:
```
from sklearn.model_selection import RandomizedSearchCV
randomRF = RandomizedSearchCV(estimator = rf,param_distributions = random_grid,
               n_iter = 100, cv = 5, verbose=2, random_state=50, n_jobs = -1)
```

In [52]:
```
randomRF.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[52]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=50),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      120],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 3, 4],
                                        'min_samples_split': [2, 6, 10],
                                        'n_estimators': [5, 20, 50, 100]},
                   random_state=50, verbose=2)
```

# Hyperparameter

```
In [53]: y_pred = class_forest.predict(x_test)

In [54]: print(x_test)

[[ 0.51868193 -1.09482178 -1.48705256  1.22568844  0.70140464]
 [-1.12909595 -0.08102359  1.49509682 -0.35054133  0.70140464]
 [ 0.24405229  0.1442649  -0.68416619 -0.63712857 -1.71151938]
 ...
 [ 0.79331158  0.25690914 -1.02826035 -1.64018388  0.70140464]
 [ 0.24405229  0.70748612  0.00402213 -2.35665195 -1.71151938]
 [-0.8544663  -0.30631207 -0.22537398 -0.20724772  0.70140464]]

In [55]: y_pred = pd.DataFrame(y_pred)
```

# Hyperparameter

In [57]:
```
print ('Random grid: ', random_grid, '\n')
```

```
Random grid:  {'n_estimators': [5, 20, 50, 100], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 5
0, 60, 70, 80, 90, 100, 110, 120], 'min_samples_split': [2, 6, 10], 'min_samples_leaf': [1, 3, 4], 'bootstrap': [Tr
ue, False]}
```

In [58]:
```
print ('Best Parameters: ', randomRF.best_params_, ' \n')
```

```
Best Parameters:  {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 3, 'max_features': 'sqrt', 'max
_depth': 20, 'bootstrap': True}
```

In [59]:
```
bestRand = RandomForestRegressor(n_estimators = 100, min_samples_split = 6, min_samples_leaf= 4,
                                 max_features = 'sqrt', max_depth= 120, bootstrap=False)
bestRand.fit( x_train, y_train)
```

Out[59]:
```
RandomForestRegressor(bootstrap=False, max_depth=120, max_features='sqrt',
                      min_samples_leaf=4, min_samples_split=6)
```

# Model Evaluation

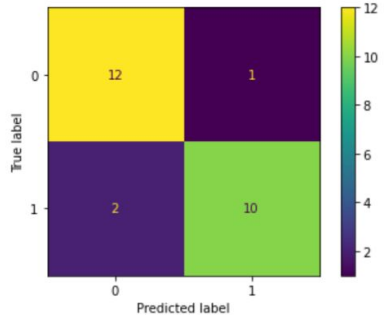
In [64]:
```
from sklearn.datasets import make_classification
x, y = make_classification(random_state=0)
x_train, x_test, y_train, y_test = train_test_split( x,y , random_state=0)
```

In [65]:
```
from sklearn.svm import SVC
clf = SVC(random_state=0)
```

In [66]:
```
clf.fit(x_train, y_train)
```

Out[66]:
```
SVC(random_state=0)
```

In [67]:
```
plot_confusion_matrix(clf, x_test, y_test)
plt.show()
```



# Model Evaluation

In [35]:
```
from sklearn.metrics import precision_score, recall_score, accuracy_score
```

In [36]:
```
print("Precision Score : ",precision_score(y_test, y_pred,
                                           pos_label='positive',
                                           average='micro'))
print("Recall Score : ",recall_score(y_test, y_pred,
                                     pos_label='positive',
                                     average='micro'))
```

```
Precision Score :  0.6494127095673126
Recall Score :  0.6494127095673126
```

In [37]:
```
print('Accuracy Score: %.3f' % accuracy_score(y_test, y_pred))
```

```
Accuracy Score: 0.649
```

# Pipeline

In [70]:
```
x_test= pd.DataFrame(x_test)
```

In [71]:
```
x_train= pd.DataFrame(x_train)
```

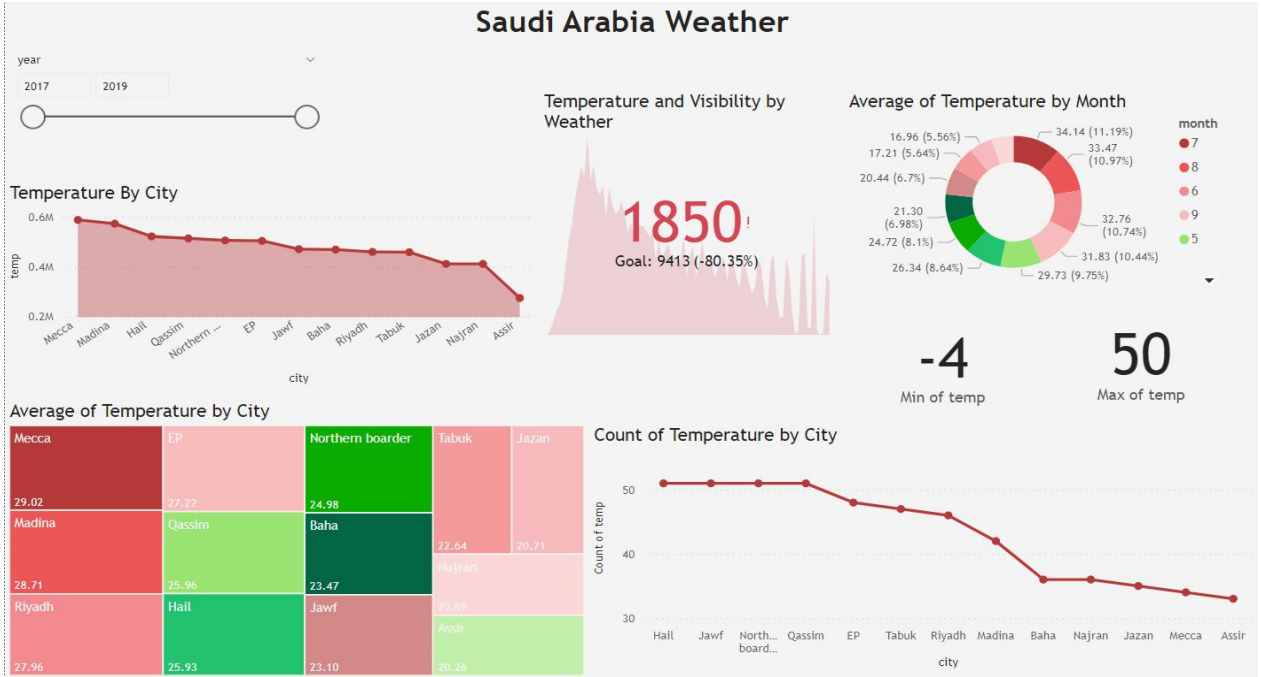
In [72]:
```
x_train_n = x_train.select_dtypes(exclude=["category", "object"])
x_test_n = x_test.select_dtypes(exclude=["category", "object"])

pipe = make_pipeline(
    SimpleImputer(),
    StandardScaler(),
    LogisticRegression()
)

pipe.fit(x_train_n,y_train)
pipe.score(x_test_n, y_test)
print(f'Pipeline Accuracy: {pipe.score(x_test_n, y_test)}')
```

```
Pipeline Accuracy: 0.92
```

# PowerBi Dashboard



# Thanks

# Resources

- weather-sa dataset
-