Importing Necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
```

Reading database and assigning to a variable

In [2]:
```
df_beijing=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Beijing_labeled.csv")
df_chengdu=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Chengdu_labeled.csv")
df_guangzhou=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Guangzhou_labeled.csv")
df_shanghai=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Shanghai_labeled.csv")
df_shenyang=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Shenyang_labeled.csv")
```

In [3]:
```
df_beijing.info() # zero null values
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2071 entries, 0 to 2070
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   season         2071 non-null   int64
 1   DEWP           2071 non-null   float64
 2   HUMI           2071 non-null   float64
 3   PRES           2071 non-null   float64
 4   TEMP           2071 non-null   float64
 5   Iws            2071 non-null   float64
 6   precipitation  2071 non-null   float64
 7   cbwd_NE        2071 non-null   int64
 8   cbwd_NW        2071 non-null   int64
 9   cbwd_SE        2071 non-null   int64
 10  PM_HIGH        2071 non-null   float64
dtypes: float64(7), int64(4)
memory usage: 178.1 KB
```

In [4]:
```
df_shenyang.info() # zero null values
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   season         824 non-null    int64
 1   DEWP           824 non-null    float64
 2   HUMI           824 non-null    float64
 3   PRES           824 non-null    float64
 4   TEMP           824 non-null    float64
 5   Iws            824 non-null    float64
 6   precipitation  824 non-null    float64
 7   cbwd_NE        824 non-null    int64
 8   cbwd_NW        824 non-null    int64
 9   cbwd_SE        824 non-null    int64
 10  PM_HIGH        824 non-null    float64
dtypes: float64(7), int64(4)
memory usage: 70.9 KB
```

In [5]:
```
df_guangzhou.info() # zero null values
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1352 entries, 0 to 1351
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   season         1352 non-null   float64
 1   DEWP           1352 non-null   float64
 2   HUMI           1352 non-null   float64
 3   PRES           1352 non-null   float64
 4   TEMP           1352 non-null   float64
 5   Iws            1352 non-null   float64
 6   precipitation  1352 non-null   float64
 7   cbwd_NE        1352 non-null   int64
 8   cbwd_NW        1352 non-null   int64
 9   cbwd_SE        1352 non-null   int64
 10  PM_HIGH        1352 non-null   float64
dtypes: float64(8), int64(3)
memory usage: 116.3 KB
```

In [6]:
```
df_shanghai.info() # zero null values
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1351 entries, 0 to 1350
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   season         1351 non-null   int64
 1   DEWP           1351 non-null   float64
 2   HUMI           1351 non-null   float64
 3   PRES           1351 non-null   float64
 4   TEMP           1351 non-null   float64
 5   Iws            1351 non-null   float64
 6   precipitation  1351 non-null   float64
 7   cbwd_NE        1351 non-null   int64
 8   cbwd_NW        1351 non-null   int64
 9   cbwd_SE        1351 non-null   int64
 10  PM_HIGH        1351 non-null   float64
dtypes: float64(7), int64(4)
memory usage: 116.2 KB
```

In [7]:
```
df_beijing.head()
```

Out[7]:

	season	DEWP	HUMI	PRES	TEMP	Iws	precipitation	cbwd_NE	cbwd_NW	cbwd_SE	PM_HIGH
0	4	-8.0	79.0	1026.0	-5.0	23.69	0.0	0	0	1	1.0
1	4	-11.0	85.0	1021.0	-9.0	105.93	1.1	0	0	1	0.0
2	4	-21.0	43.0	1030.0	-11.0	117.55	0.0	0	1	0	0.0
3	4	-25.0	33.0	1034.0	-12.0	39.35	0.0	1	0	0	0.0
4	4	-24.0	30.0	1034.0	-10.0	59.00	0.0	1	0	0	0.0

In [8]:
```
df_beijing.describe()
```

Out[8]:

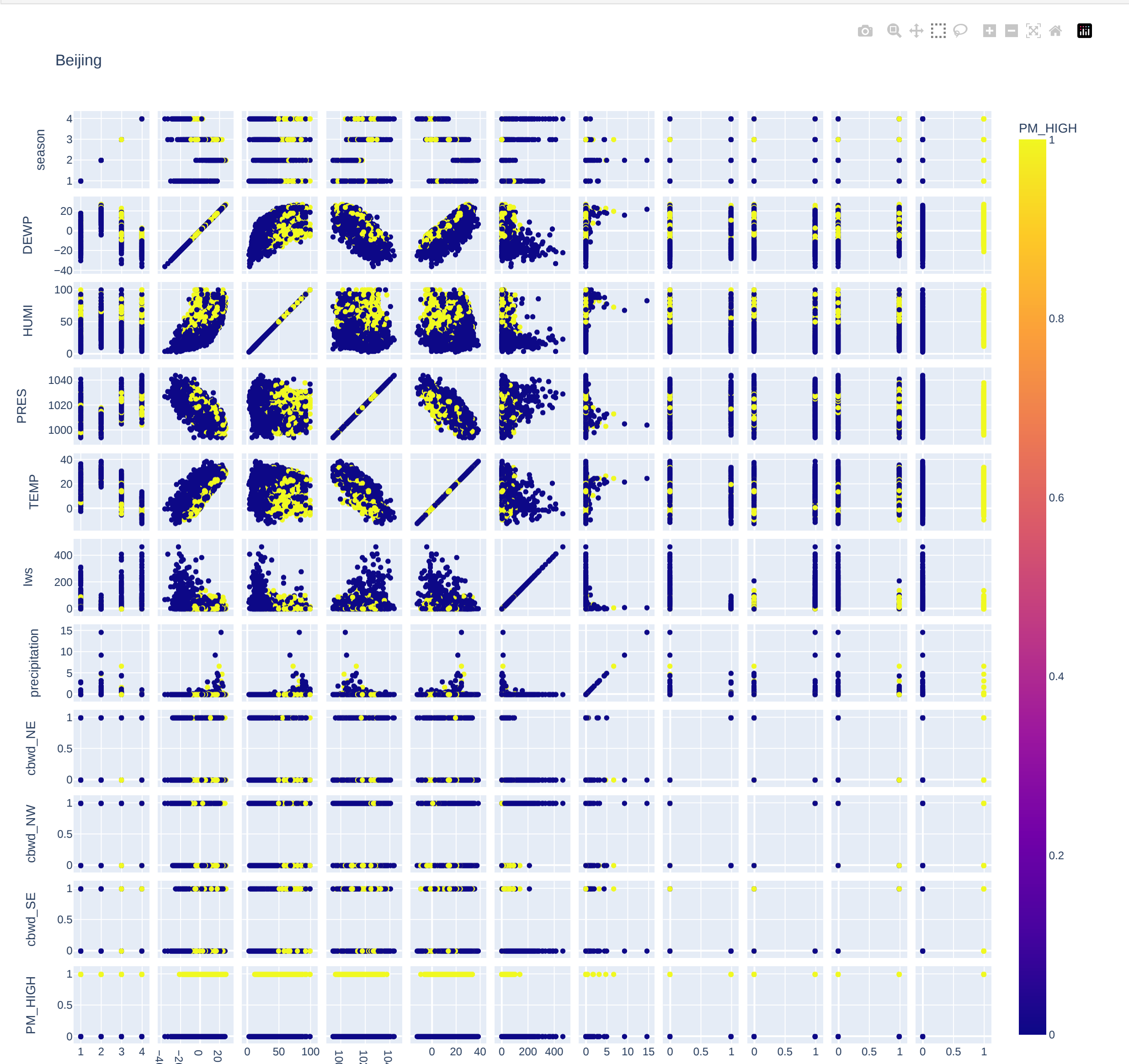
	season	DEWP	HUMI	PRES	TEMP	Iws	precipitation	cbwd_NE	cbwd_NW	cbwd_SE	PM_HIGH
count	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000
mean	2.485756	1.201835	41.121197	1016.711733	16.220666	21.499034	0.046548	0.113955	0.306132	0.315789	0.313375
std	1.123601	14.617897	22.833175	10.255643	11.688141	47.630452	0.495941	0.317833	0.460997	0.464942	0.463978
min	1.000000	-36.000000	3.000000	994.000000	-12.000000	0.450000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	-11.000000	22.000000	1008.000000	5.000000	1.790000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	2.000000	1.000000	37.000000	1016.000000	18.000000	4.470000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	3.500000	15.000000	57.000000	1025.000000	27.000000	16.990000	0.000000	0.000000	1.000000	1.000000	1.000000
max	4.000000	27.000000	100.000000	1044.000000	39.000000	468.050000	14.700000	1.000000	1.000000	1.000000	1.000000

In [9]:
```
df_beijing.shape
```

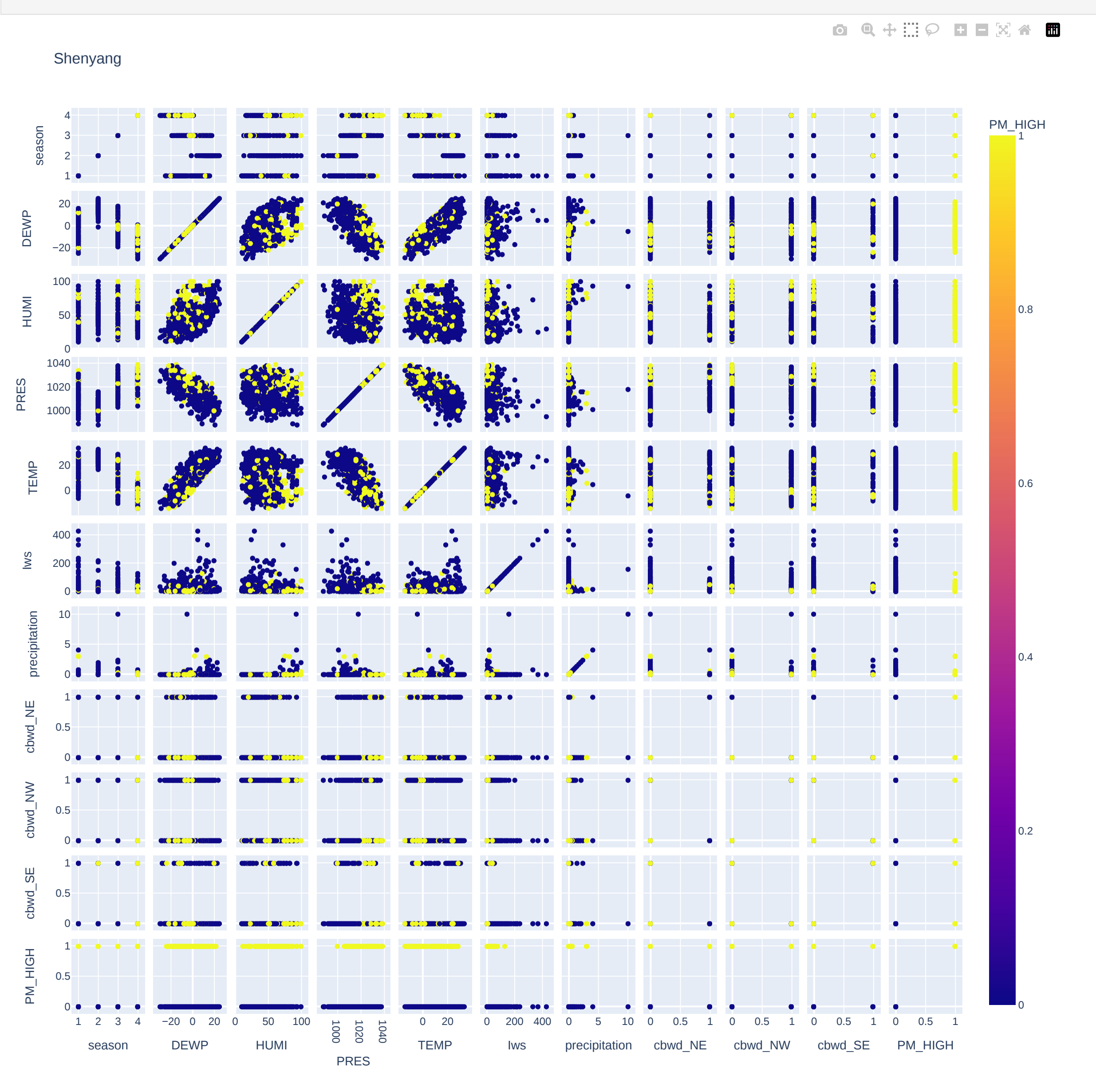
Out[9]: (2071, 11)

scatter matrix (showing relation between each variable)

In [10]:
```
import plotly.express as px
fig = px.scatter_matrix(df_beijing,color = 'PM_HIGH',  title="Beijing")
fig.update_layout(
    width=1200,
    height=1200
)
fig.show()
```



In [11]:
```
import plotly.express as px
fig = px.scatter_matrix(df_shenyang,color = 'PM_HIGH',  title="Shenyang")
fig.update_layout(
    width=1200,
    height=1200
)
fig.show()
```



Creating Class and required functions for KNN model

Implementation:

1. Calculate the distance between a test point and every point in the training set. We are considering Euclidean distance here.
2. Sort the distances and pick the k nearest distances from the list. These k nearest distances correspond to the k nearest neighbors to the test point.
3. Find the labels for these k nearest neighbors and the most common label among these neighbors is decided as the label for the test point.

In [12]:
```
# finding the most common occuring element
def most_common(lst):
    return max(set(lst), key=lst.count)
```

In [13]:
```
# Euclidean distance between test point & data
def euclidean(point, data):
    return np.sqrt(np.sum((point - data)**2, axis=1))
```

In [14]:
```
class KNNC:
    def __init__(self, k, dist_metric=euclidean):
        self.k = k
        self.dist_metric = dist_metric

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        neighbors = []
        for x in X_test:
            distances = self.dist_metric(x, self.X_train)
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
            neighbors.append(y_sorted[:self.k])
        return list(map(most_common, neighbors))

    def evaluate(self, X_test, y_test):
        y_pred = self.predict(X_test)
        accuracy = sum(y_pred == y_test) / len(y_test)
        return accuracy
```

Creating datframes to input in our model

In [15]:
```
X_beijing = df_beijing.iloc[:, :10]
y_beijing = df_beijing.iloc[:,-1]
X_shenyang = df_shenyang.iloc[:, :10]
y_shenyang = df_shenyang.iloc[:,-1]
```

In [16]:
```
# X_beijing.head()
# y_beijing.head()
# X_shenyang.head()
# y_shenyang.head()
# X_beijing.shape
```

Splitting the datasets into train and validation sets

In [17]:
```
X_beijing_train, X_beijing_validation, y_beijing_train, y_beijing_validation = train_test_split( X_beijing, y_beijing, test_size=0.2, random_state=1)
X_shenyang_train, X_shenyang_validation, y_shenyang_train, y_shenyang_validation = train_test_split( X_shenyang, y_shenyang, test_size=0.2, random_state=1)
```

Merging the train and test sets for the two cities

In [18]:
```
# Training
X_beijing_shenyang_train = pd.concat([X_beijing_train, X_shenyang_train],ignore_index = True, sort = False)
y_beijing_shenyang_train = pd.concat([y_beijing_train, y_shenyang_train],ignore_index = True, sort = False)
# X_beijing_shenyang_train.head()
```

In [19]:
```
# Test
X_beijing_shenyang_validation = pd.concat([X_beijing_validation, X_shenyang_validation],ignore_index = True, sort = False)
y_beijing_shenyang_validation = pd.concat([y_beijing_validation, y_shenyang_validation],ignore_index = True, sort = False)
```

In [20]:
```
# preprocessing ( scaling the training )
X_beijing_shenyang_train = preprocessing.StandardScaler().fit_transform(X_beijing_shenyang_train)
X_beijing_shenyang_validation = preprocessing.StandardScaler().fit_transform(X_beijing_shenyang_validation)
```

In [21]:
```
# X_beijing_shenyang_train.shape
# X_beijing_shenyang_validation.shape
# X_beijing_shenyang_train[0:5]
# X_beijing_shenyang_validation[0:5]
```

Checking the training accuracy of our implementation

In [22]:
```
train_accuracy_1 = []
ks = range(1, 11)
for k in ks:
    knn = KNNC(k=k)
    knn.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    accuracy = knn.evaluate(X_beijing_shenyang_train, y_beijing_shenyang_train)
    train_accuracy_1.append(accuracy)
# print(train_accuracy_1)
```

In [23]:
```
result_training = np.average(train_accuracy_1)
print('Training Accuracy', result_training)
```
```
Training Accuracy 0.8517926565874732
```

Checking the training accuracy of our Scikit learn library

In [24]:
```
train_accuracy_2 = []
ks = range(1, 11)
for k in ks:
    knnmodel = KNeighborsClassifier(n_neighbors = k)
    knnmodel.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    scikit_result_training = knnmodel.score(X_beijing_shenyang_train, y_beijing_shenyang_train)
    train_accuracy_2.append(scikit_result_training)
# print(train_accuracy_2)
print("Training Accuracy_scikit", scikit_result_training)
```
```
Training Accuracy_scikit 0.8190064794816415
```

Checking the validation accuracy of our implementation

In [25]:
```
validation_accuracy_1 = []
ks = range(1, 11)
for k in ks:
    knn = KNNC(k=k)
    knn.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    accuracy = knn.evaluate(X_beijing_shenyang_validation, y_beijing_shenyang_validation)
    validation_accuracy_1.append(accuracy)
print(validation_accuracy_1)
```
```
[0.7258620689655172, 0.7586206896551724, 0.7551724137931034, 0.7775862068965518, 0.7741379310344828, 0.7862068965517242, 0.8, 0.7844827586206896, 0.7793103448275862, 0.787931034482
7586]
```

In [26]:
```
result_validation = np.average(validation_accuracy_1)
print('Validation Accuracy', result_validation)
```
```
Validation Accuracy 0.7729310344827585
```

Checking the validation accuracy of our Scikit learn library

In [27]:
```
validation_accuracy_2 = []
ks = range(1, 11)
for k in ks:
    knnmodel = KNeighborsClassifier(n_neighbors = k)
    knnmodel.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    scikit_result_validation = knnmodel.score(X_beijing_shenyang_validation, y_beijing_shenyang_validation)
    validation_accuracy_2.append(scikit_result_validation)
print(validation_accuracy_2)
print('Validation Accuracy_scikit', scikit_result_validation)
```
```
[0.7258620689655172, 0.7586206896551724, 0.7551724137931034, 0.7775862068965518, 0.7741379310344828, 0.7862068965517242, 0.8, 0.7844827586206896, 0.7793103448275862, 0.787931034482
7586]
Validation Accuracy_scikit 0.7879310344827586
```

plotting training accuracy for our implementation and scikit learn

In [28]:
```
import plotly.graph_objects as go
n = np.linspace(1, 10, num=10, dtype=int)
fig = go.Figure()
fig.update_xaxes(title_text="Number of neighbors(k)")
fig.update_yaxes(title_text="Accuracy")
fig.update_traces(textposition='top center')
fig.update_layout(title_text='Accuracy vs k (for training dataset)',
                  title_x=0.5,
                  height = 400)
fig.add_traces(go.Scatter(x= n , y= train_accuracy_1, name='own implementation'))
fig.add_traces(go.Scatter(x= n, y= train_accuracy_2, name='scikit learn model'))
fig.show()
```

plotting validation accuracy for our implementation and scikit learn

In [29]:
```
import plotly.graph_objects as go
n = np.linspace(1, 10, num=10, dtype=int)
fig = go.Figure()
fig.update_xaxes(title_text="Number of neighbors(k)")
fig.update_yaxes(title_text="Accuracy")
fig.update_traces(textposition='top center')
fig.update_layout(title_text='Accuracy vs k (for training dataset)',
                  title_x=0.5,
                  height = 400)
fig.add_traces(go.Scatter(x= n , y= validation_accuracy_1, name='own implementation'))
fig.add_traces(go.Scatter(x= n, y= validation_accuracy_2, name='scikit learn model'))
fig.show()
```

Testing the model for Guanzhou and Shanghai

In [30]:
```
# Guanzhou
X_guanzhou = df_guangzhou.iloc[:, :10]
y_guanzhou = df_guangzhou.iloc[:,-1]

# Shanghai
X_shanghai = df_shanghai.iloc[:, :10]
y_shanghai = df_shanghai.iloc[:,-1]

# X_guanzhou.head()
```

In [31]:
```
# preprocessing
X_guanzhou = preprocessing.StandardScaler().fit_transform(X_guanzhou)
X_shanghai = preprocessing.StandardScaler().fit_transform(X_shanghai)
```

Checking the accuracy for Guanzhou with our implementation

In [32]:
```
accuracy_guanzhou = []
ks = range(1, 11)
for k in ks:
    knn = KNNC(k=k)
    knn.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    accuracy = knn.evaluate(X_guanzhou, y_guanzhou)
    accuracy_guanzhou.append(accuracy)
print(accuracy_guanzhou)
```
```
[0.6967455621301775, 0.8180473372781065, 0.7204142011834319, 0.7892011834319527, 0.7374260355029586, 0.7788461538461539, 0.7477810650887574, 0.783284023668639, 0.7440828402366864,
0.7803254437869822]
```

In [33]:
```
result_guanzhou = np.average(accuracy_guanzhou)
print('Validation Accuracy', result_guanzhou)
```
```
Validation Accuracy 0.7596153846153847
```

Checking the accuracy for Guanzhou with scikit learn

In [34]:
```
accuracy_guanzhou_scikit = []
ks = range(1, 11)
for k in ks:
    knnmodel = KNeighborsClassifier(n_neighbors = k)
    knnmodel.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    scikit_result_validation = knnmodel.score(X_guanzhou, y_guanzhou)
    accuracy_guanzhou_scikit.append(scikit_result_validation)
# print(accuracy_guanzhou_scikit)
print('accuracy_guanzhou_scikit', scikit_result_validation)
```
```
accuracy_guanzhou_scikit 0.7803254437869822
```

plotting accuracy vs K for our implementation and scikit learn (Guamzhou)

In [35]:
```
import plotly.graph_objects as go
n = np.linspace(1, 10, num=10, dtype=int)
fig = go.Figure()
fig.update_xaxes(title_text="Number of neighbors(k)")
fig.update_yaxes(title_text="Accuracy")
fig.update_traces(textposition='top center')
fig.update_layout(title_text='Accuracy vs k (Guanzhou)',
                  title_x=0.5,
                  height = 400)
fig.add_traces(go.Scatter(x= n , y= accuracy_guanzhou, name='own implementation'))
fig.add_traces(go.Scatter(x= n, y= accuracy_guanzhou_scikit, name='scikit learn model'))
fig.show()
```

Checking the accuracy for Shanghai with our implementation

In [36]:
```
accuracy_shanghai = []
ks = range(1, 11)
for k in ks:
    knn = KNNC(k=k)
    knn.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    accuracy = knn.evaluate(X_shanghai, y_shanghai)
    accuracy_shanghai.append(accuracy)
print(accuracy_shanghai)
```
```
[0.694300518134715, 0.8134715025906736, 0.7261287934863064, 0.8045891931902295, 0.7327905255366395, 0.7823834196891192, 0.7335307179866766, 0.7772020725388601, 0.7298297557364914,
0.764618800888231]
```

In [37]:
```
result_shangahi = np.average(accuracy_shanghai)
print('Validation Accuracy', result_shangahi)
```
```
Validation Accuracy 0.7558845299777943
```

Checking the accuracy for Shanghai with scikit learn

In [38]:
```
accuracy_shanghai_scikit = []
ks = range(1, 11)
for k in ks:
    knnmodel = KNeighborsClassifier(n_neighbors = k)
    knnmodel.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    scikit_result_validation = knnmodel.score(X_shanghai, y_shanghai)
    accuracy_shanghai_scikit.append(scikit_result_validation)
# print(accuracy_shanghai_scikit)
print('accuracy_shanghai_scikit', scikit_result_validation)
```
```
accuracy_shanghai_scikit 0.764618800888231
```

plotting accuracy vs K for our implementation and scikit learn (Shanghai)

In [39]:
```
import plotly.graph_objects as go
n = np.linspace(1, 10, num=10, dtype=int)
fig = go.Figure()
fig.update_xaxes(title_text="Number of neighbors(k)")
fig.update_yaxes(title_text="Accuracy")
fig.update_traces(textposition='top center')
fig.update_layout(title_text='Accuracy vs k for Shanghai',
                  title_x=0.5,
                  height = 400)
fig.add_traces(go.Scatter(x= n , y= accuracy_shanghai, name='own implementation'))
fig.add_traces(go.Scatter(x= n, y= accuracy_shanghai_scikit, name='scikit learn model'))
fig.show()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: