Overview

This notebook will show you how to create and query a table or DataFrame that you uploaded to DBFS. DBFS (https://docs.databricks.com/user-guide/dbfs-databricks-file-system.html) is a Databricks File System that allows you to store data for querying inside of Databricks. This notebook assumes that you have a file already inside of DBFS that you would like to read from.

This notebook is written in **Python** so the default cell type is Python. However, you can use different languages by using the `%LANGUAGE` syntax. Python, Scala, SQL, and R are all supported.

Project Topic: Predicting the thermal comfort by using time series data

Business Problem: The project aims to predict individual thermal comfort levels based on multiple environmental factors. The intention is to build a predictive time series model and help HVAC (building manager) to optimize energy usage while ensuring occupant comfort.

1. Import Libraries

Import the required libraries

5

```
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, mean, stddev
from pyspark.ml.feature import Imputer, VectorAssembler, StandardScaler
from pyspark.ml import Pipeline
import numpy as np

import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classificat
import warnings
```

2. Load the Data

7

```
# File location and type
file_location = "/FileStore/tables/ashrae_db2_01-1.csv"
file_type = "csv"

# CSV options
infer_schema = "false"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df.head(10))
```

Table


```
# Assuming 'spark_df' is your PySpark DataFrame
pandas_df = df.toPandas()

# Now you can work with your pandas DataFrame (pandas_df)
(pandas_df.head(3))
```

	Publication (Citation)	Data contributor	Year	Season	Koppen climate classification	Climate	City	Country	Building type	Cooling startegy_building level	Coolin startegy_operatio mode for MI building
0	"Kwok, Alison G., and Chungyoon Chun. ""Therma...	Kwok Alison	2000	Summer	Cfa	Humid subtropical	Tokyo	Japan	Classroom	Air Conditioned	N
1	"Kwok, Alison G., and Chungyoon Chun. ""Therma...	Kwok Alison	2000	Summer	Cfa	Humid subtropical	Tokyo	Japan	Classroom	Air Conditioned	N
2	"Kwok, Alison G., and Chungyoon Chun. ""Therma...	Kwok Alison	2000	Summer	Cfa	Humid subtropical	Tokyo	Japan	Classroom	Air Conditioned	N

3. Data Exploration

Fing the total quanity of the data, missing data, na values, outliers, maximum, minimum

```
def explore_data(df):
    results = []
    for col in df.columns:
        # Count missing values (NaN)
        missing_count = df[col].isnull().sum()
        # Count NA values (as a string)
        na_count = df[col].astype(str).str.contains('NA').sum()
        # Count total values
        total_count = len(df[col])
        # Calculate percentage of missing and NA values
        missing_percent = (missing_count / total_count) * 100
        na_percent = (na_count / total_count) * 100
        # Determine data type
        data_type = df[col].dtype

# --- Additional Checks ---

        # Unique values
        unique_count = df[col].nunique()

        # Detect outliers (using IQR method)
        if pd.api.types.is_numeric_dtype(df[col]):  # Only for numeric
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            outlier_count = ((df[col] < lower_bound) | (df[col] > upper_bound)).sum()
        else:
            outlier_count = np.nan  # Not applicable for non-numeric

        # Minimum and Maximum values
        if pd.api.types.is_numeric_dtype(df[col]):
            minimum = df[col].min()
            maximum = df[col].max()
        else:
            minimum = np.nan
            maximum = np.nan

        results.append([
            col, total_count, missing_count, missing_percent,
            na_count, na_percent, data_type, unique_count,
            outlier_count, minimum, maximum
        ])

    exploration_df = pd.DataFrame(results, columns=[
        "Column", "Total Count", "Missing Count", "Missing (%)",
        "NA Count", "NA (%)", "Data Type", "Unique Values",
        "Outlier Count", "Minimum", "Maximum"
    ])
    return exploration_df

# Perform data exploration
exploration_results = explore_data(pandas_df)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
exploration_results.head(100)
```

	Column	Total Count	Missing Count	Missing (%)	NA Count	NA (%)	Data Type	Unique Values	Outlier Count	Minimum	Maximum
0	Publication (Citation)	107583	0	0.0	1655	1.538347	object	67	NaN	NaN	NaN
1	Data contributor	107583	0	0.0	0	0.000000	object	58	NaN	NaN	NaN
2	Year	107583	0	0.0	0	0.000000	object	39	NaN	NaN	NaN
3	Season	107583	0	0.0	241	0.224013	object	16	NaN	NaN	NaN
4	Koppen climate classification	107583	0	0.0	248	0.230520	object	30	NaN	NaN	NaN
5	Climate	107583	0	0.0	0	0.000000	object	28	NaN	NaN	NaN
6	City	107583	0	0.0	13	0.012084	object	99	NaN	NaN	NaN
7	Country	107583	0	0.0	0	0.000000	object	45	NaN	NaN	NaN
8	Building type	107583	0	0.0	4199	3.903033	object	22	NaN	NaN	NaN
9	Cooling startegy_building level	107583	0	0.0	1227	1.140515	object	12	NaN	NaN	NaN
10	Cooling startegy_operation mode for MM buildings	107583	0	0.0	64100	59.581904	object	10	NaN	NaN	NaN
11	Heating strategy_building level	107583	0	0.0	56049	52.098380	object	7	NaN	NaN	NaN
12	Age	107583	0	0.0	68076	63.277655	object	99	NaN	NaN	NaN
13	Sex	107583	0	0.0	55393	51.488618	object	52	NaN	NaN	NaN
14	Thermal sensation	107583	0	0.0	7041	6.544714	object	109	NaN	NaN	NaN
15	Thermal sensation acceptability	107583	0	0.0	40390	37.543106	object	102	NaN	NaN	NaN
16	Thermal preference	107583	0	0.0	24063	22.366917	object	117	NaN	NaN	NaN
17	Air movement acceptability	107583	0	0.0	81374	75.638344	object	66	NaN	NaN	NaN
18	Air movement preference	107583	0	0.0	65421	60.809793	object	13	NaN	NaN	NaN
19	Thermal comfort	107583	0	0.0	67781	63.003449	object	32	NaN	NaN	NaN
20	PMV	107583	0	0.0	41217	38.311815	object	614	NaN	NaN	NaN
21	PPD	107583	0	0.0	36415	33.848285	object	5376	NaN	NaN	NaN
22	SET	107583	0	0.0	36139	33.591738	object	3857	NaN	NaN	NaN
23	Clo	107583	0	0.0	18007	16.737775	object	3533	NaN	NaN	NaN
24	Met	107583	0	0.0	25766	23.949881	object	1966	NaN	NaN	NaN
25	activity_10	107583	0	0.0	87522	81.353002	object	782	NaN	NaN	NaN
26	activity_20	107583	0	0.0	93444	86.857589	object	202	NaN	NaN	NaN
27	activity_30	107583	0	0.0	93726	87.119712	object	188	NaN	NaN	NaN
28	activity_60	107583	0	0.0	92899	86.351003	object	92	NaN	NaN	NaN
29	Air temperature (C)	107583	0	0.0	22703	21.102776	object	402	NaN	NaN	NaN
30	Air temperature (F)	107583	0	0.0	13605	12.646050	object	816	NaN	NaN	NaN
31	Ta_h (C)	107583	0	0.0	72457	67.349860	object	657	NaN	NaN	NaN
32	Ta_h (F)	107583	0	0.0	79000	73.431676	object	508	NaN	NaN	NaN
33	Ta_m (C)	107583	0	0.0	71698	66.644358	object	498	NaN	NaN	NaN
34	Ta_m (F)	107583	0	0.0	71941	66.870230	object	592	NaN	NaN	NaN
35	Ta_l (C)	107583	0	0.0	90810	84.409247	object	534	NaN	NaN	NaN
36	Ta_l (F)	107583	0	0.0	93032	86.474629	object	437	NaN	NaN	NaN
37	Operative temperature (C)	107583	0	0.0	79489	73.886209	object	484	NaN	NaN	NaN
38	Operative temperature (F)	107583	0	0.0	77267	71.820827	object	698	NaN	NaN	NaN
39	Radiant temperature (C)	107583	0	0.0	75228	69.925546	object	615	NaN	NaN	NaN
40	Radiant temperature (F)	107583	0	0.0	71566	66.521662	object	733	NaN	NaN	NaN
41	Globe temperature (C)	107583	0	0.0	85439	79.416822	object	601	NaN	NaN	NaN
42	Globe temperature (F)	107583	0	0.0	85123	79.123096	object	584	NaN	NaN	NaN
43	Tg_h (C)	107583	0	0.0	44276	41.155201	object	605	NaN	NaN	NaN
44	Tg_h (F)	107583	0	0.0	50475	46.917264	object	496	NaN	NaN	NaN
45	Tg_m (C)	107583	0	0.0	83879	77.966779	object	483	NaN	NaN	NaN
46	Tg_m (F)	107583	0	0.0	82256	76.458176	object	592	NaN	NaN	NaN
47	Tg_l (C)	107583	0	0.0	78609	73.068236	object	526	NaN	NaN	NaN

48	Tg_l (F)	107583	0	0.0	80232	74.576838	object	457	NaN	NaN	NaN
49	Relative humidity (%)	107583	0	0.0	28753	26.726342	object	837	NaN	NaN	NaN
50	Humidity preference	107583	0	0.0	91175	84.748520	object	612	NaN	NaN	NaN
51	Humidity sensation	107583	0	0.0	86054	79.988474	object	551	NaN	NaN	NaN
52	Air velocity (m/s)	107583	0	0.0	29585	27.499698	object	375	NaN	NaN	NaN
53	Air velocity (fpm)	107583	0	0.0	23149	21.517340	object	4539	NaN	NaN	NaN
54	Velocity_h (m/s)	107583	0	0.0	80193	74.540587	object	1039	NaN	NaN	NaN
55	Velocity_h (fpm)	107583	0	0.0	85044	79.049664	object	721	NaN	NaN	NaN
56	Velocity_m (m/s)	107583	0	0.0	76359	70.976827	object	534	NaN	NaN	NaN
57	Velocity_m (fpm)	107583	0	0.0	74787	69.515630	object	2524	NaN	NaN	NaN
58	Velocity_l (m/s)	107583	0	0.0	91655	85.194687	object	637	NaN	NaN	NaN
59	Velocity_l (fpm)	107583	0	0.0	93227	86.655884	object	458	NaN	NaN	NaN
60	Subject«s height (cm)	107583	0	0.0	89709	83.385851	object	431	NaN	NaN	NaN
61	Subject«s weight (kg)	107583	0	0.0	86614	80.509002	object	204	NaN	NaN	NaN
62	Blind (curtain)	107583	0	0.0	100616	93.524070	object	236	NaN	NaN	NaN
63	Fan	107583	0	0.0	94397	87.743417	object	3	NaN	NaN	NaN
64	Window	107583	0	0.0	85504	79.477241	object	51	NaN	NaN	NaN
65	Door	107583	0	0.0	91417	84.973462	object	71	NaN	NaN	NaN
66	Heater	107583	0	0.0	99245	92.249705	object	3	NaN	NaN	NaN

4. Data Preparation

Removing unwanted columns, converting categorical data

Selecting below columns for anlaysis, removing columns with high number of NAs:

'Air temperature (C)', 'Air velocity (m/s)', 'Relative humidity (%)', 'Clo', 'Met', 'Season', 'Koppen climate classification', 'Building type', 'Cooling startegy_building level', 'Outdoor monthly air temperature (C)', 'Thermal comfort'

12

```
select_columns = ['Air temperature (C)', 'Air velocity (m/s)', 'Relative humidity (%)', 'Clo', 'Met',
            'Season', 'Koppen climate classification', 'Building type', 'Cooling startegy_building level',
'Outdoor monthly air temperature (C)', 'Thermal comfort']

data_df = pandas_df[select_columns]

data_df.describe()
```

	Air temperature (C)	Air velocity (m/s)	Relative humidity (%)	Clo	Met	Season	Koppen climate classification	Building type	Cooling startegy_building level	Outdoor monthly air temperature (C)	Thermal comfort
count	107583	107583	107583	107583	107583	107583	107583	107583	107583	107583	107583
unique	402	375	837	3533	1966	16	30	22	12	432	32
top	NA	NA	NA	NA	NA	Winter	Csa	Office	Naturally Ventilated	NA	NA

13

```
# Drop NA records
data_df.replace(['NA', 'Na'], pd.NA, inplace=True)
#data_df['Thermal comfort'].replace('Na', pd.NA, inplace=True)
data_df = data_df.dropna(how='any')
```

```
/root/.ipykernel/1356/command-1519290717910526-3418513481:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data_df.replace(['NA', 'Na'], pd.NA, inplace=True)
```

14

```
#Check data after dropping NAs
exploration_results = explore_data(data_df)
exploration_results.head(100)
```

	Column	Total Count	Missing Count	Missing (%)	NA Count	NA (%)	Data Type	Unique Values	Outlier Count	Minimum	Maximum
0	Air temperature (C)	16209	0	0.0	0	0.0	object	279	NaN	NaN	NaN
1	Air velocity (m/s)	16209	0	0.0	0	0.0	object	226	NaN	NaN	NaN
2	Relative humidity (%)	16209	0	0.0	0	0.0	object	754	NaN	NaN	NaN
3	Clo	16209	0	0.0	0	0.0	object	237	NaN	NaN	NaN
4	Met	16209	0	0.0	0	0.0	object	24	NaN	NaN	NaN
5	Season	16209	0	0.0	0	0.0	object	4	NaN	NaN	NaN
6	Koppen climate classification	16209	0	0.0	0	0.0	object	10	NaN	NaN	NaN
7	Building type	16209	0	0.0	0	0.0	object	5	NaN	NaN	NaN
8	Cooling startegy_building level	16209	0	0.0	0	0.0	object	3	NaN	NaN	NaN
9	Outdoor monthly air	16209	0	0.0	0	0.0	object	295	NaN	NaN	NaN

15

```
#describe the data
data_df.describe()
```

	Air temperature (C)	Air velocity (m/s)	Relative humidity (%)	Clo	Met	Season	Koppen climate classification	Building type	Cooling startegy_building level	Outdoor monthly air temperature (C)	Thermal comfort
count	16209	16209	16209	16209	16209	16209	16209	16209	16209	16209	16209
unique	279	226	754	237	24	4	10	5	3	295	17
top	24.2	0.05	65.0	0.57	1.10	Summer	BSh	Office	Naturally Ventilated	26.0	5

16

```
#Convert the data to appropriate datatype
data_df['Air temperature (C)'] = data_df['Air temperature (C)'].astype(float)
data_df['Air velocity (m/s)'] = data_df['Air velocity (m/s)'].astype(float)
data_df['Relative humidity (%)'] = data_df['Relative humidity (%)'].astype(float)
data_df['Clo'] = data_df['Clo'].astype(float)
data_df['Outdoor monthly air temperature (C)'] = data_df['Outdoor monthly air temperature (C)'].astype(float)
data_df['Met']=data_df['Met'].astype('category')
data_df['Season']=data_df['Season'].astype('category')
data_df['Koppen climate classification']=data_df['Koppen climate classification'].astype('category')
data_df['Building type']=data_df['Building type'].astype('category')
data_df['Cooling startegy_building level']=data_df['Cooling startegy_building level'].astype('category')
data_df['Thermal comfort'] = data_df['Thermal comfort'].astype('category')
```

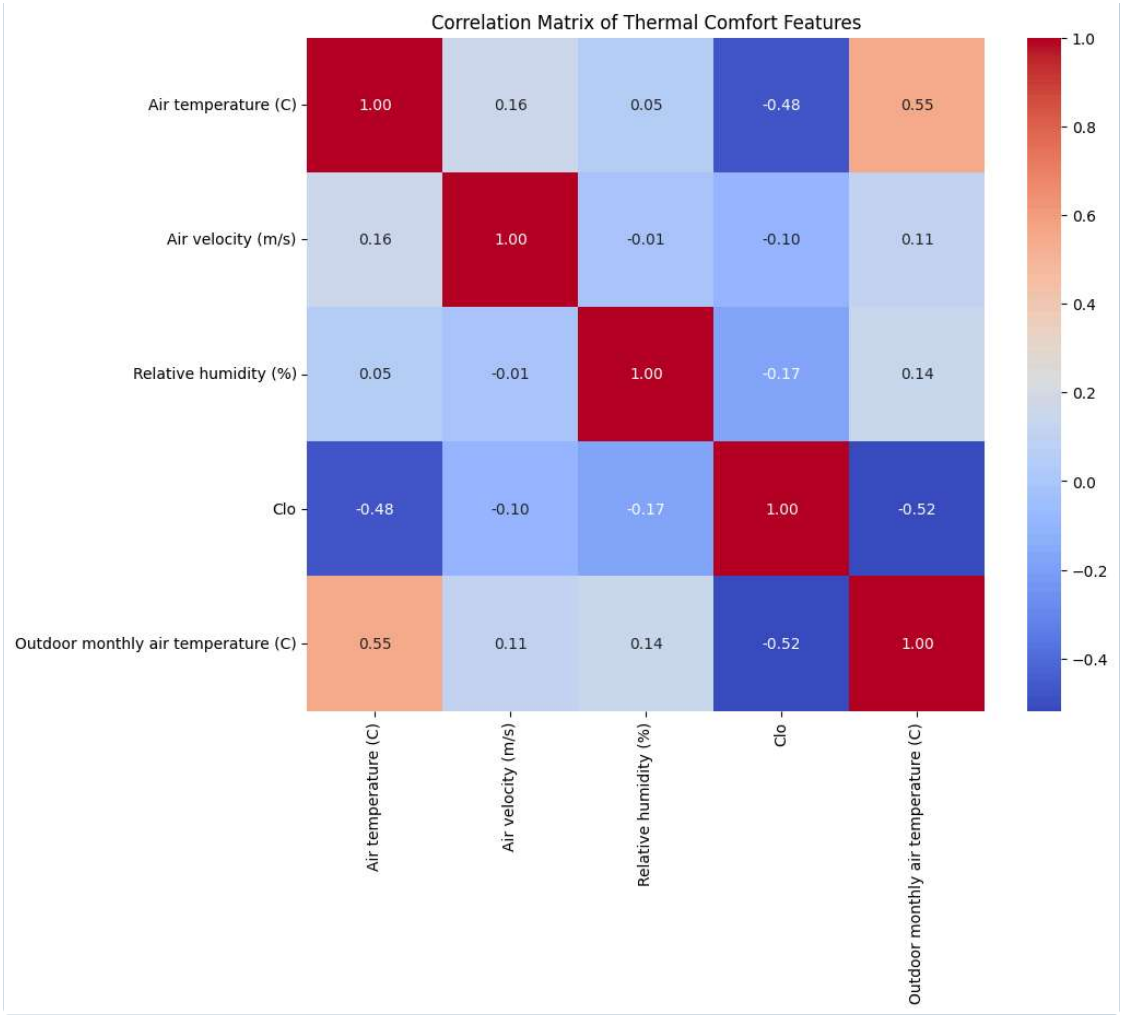
5. Visualize the Data

Correlation Matrix: Correlations between numerical features.


```
# Calculate the correlation matrix
corr_matrix = data_df.corr()

# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Thermal Comfort Features')
plt.show()
```

```
/root/.ipykernel/1356/command-1519290717910532-3861020466:2: FutureWarning: The default value of numeric_only in Dat
aFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the v
alue of numeric_only to silence this warning.
  corr_matrix = data_df.corr()
```

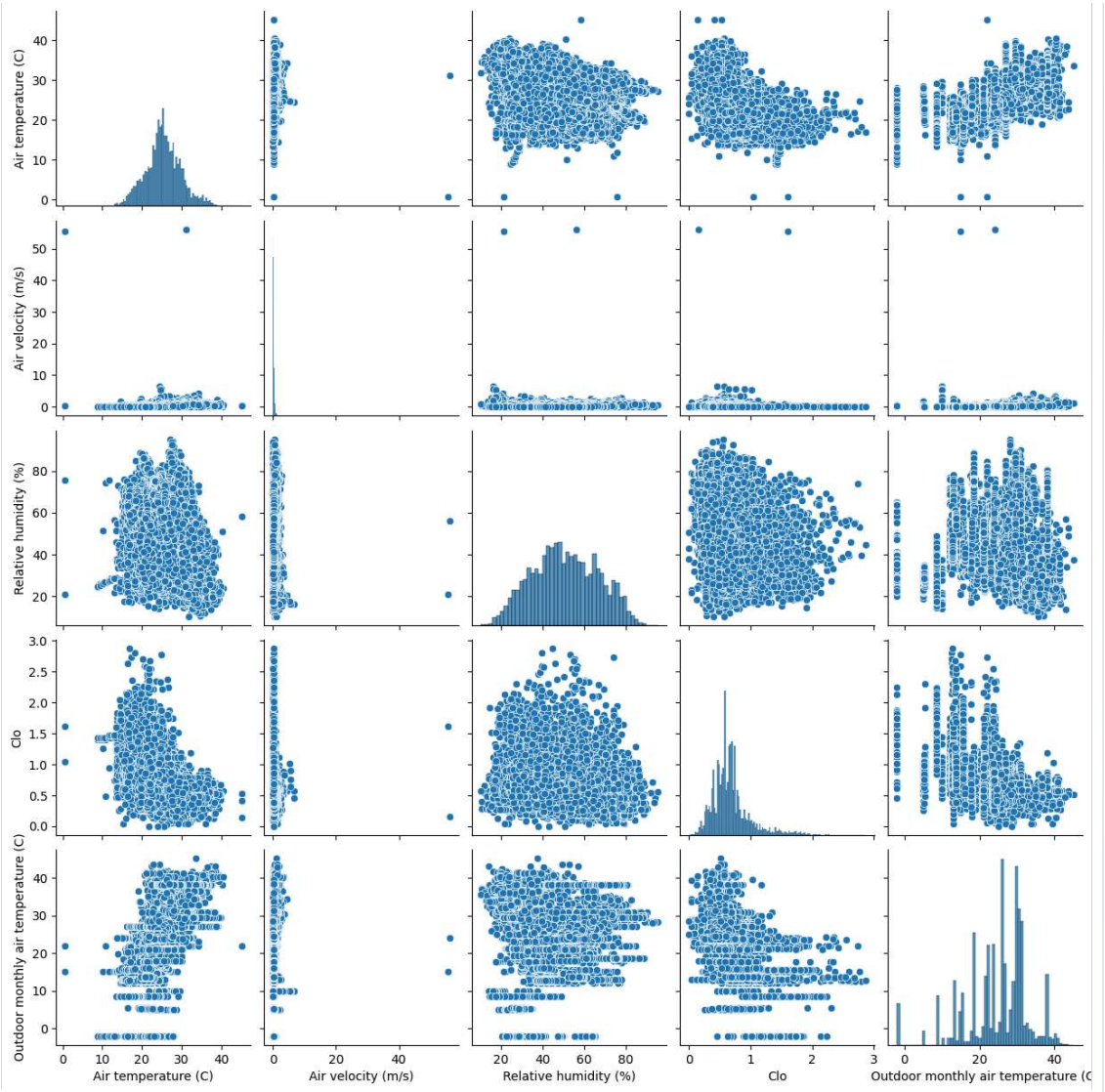


Pair Plot: To see both distributions of individual variables and their relationships

21

```
sns.pairplot(data_df)
plt.show()
```

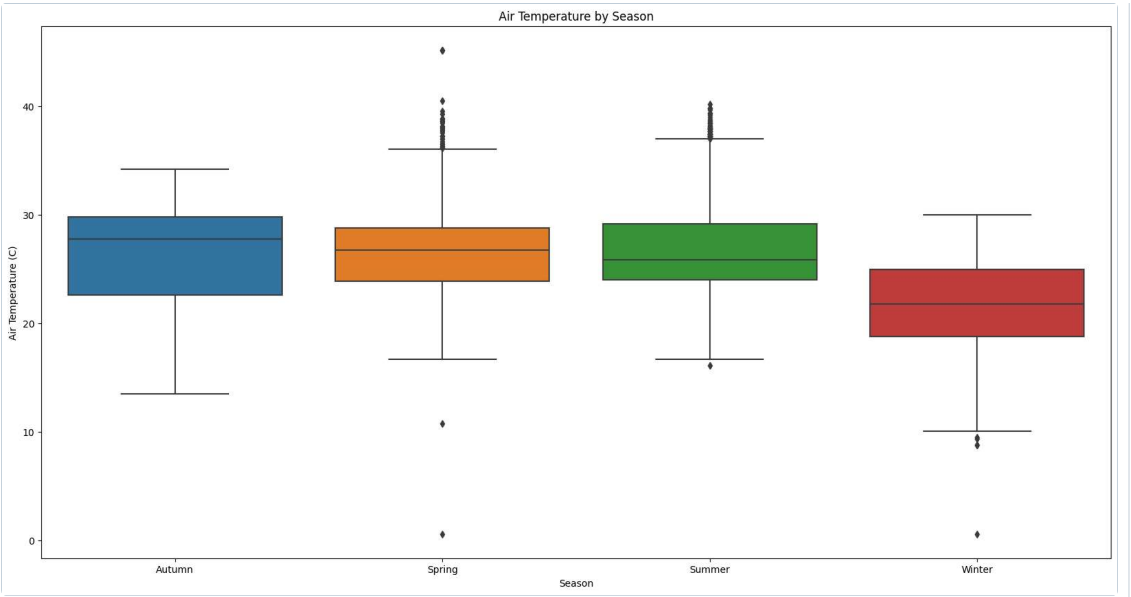
```
/databricks/python/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed
to tight
  self._figure.tight_layout(*args, **kwargs)
```



Box Plot Distribution of a numerical variable across different categories.

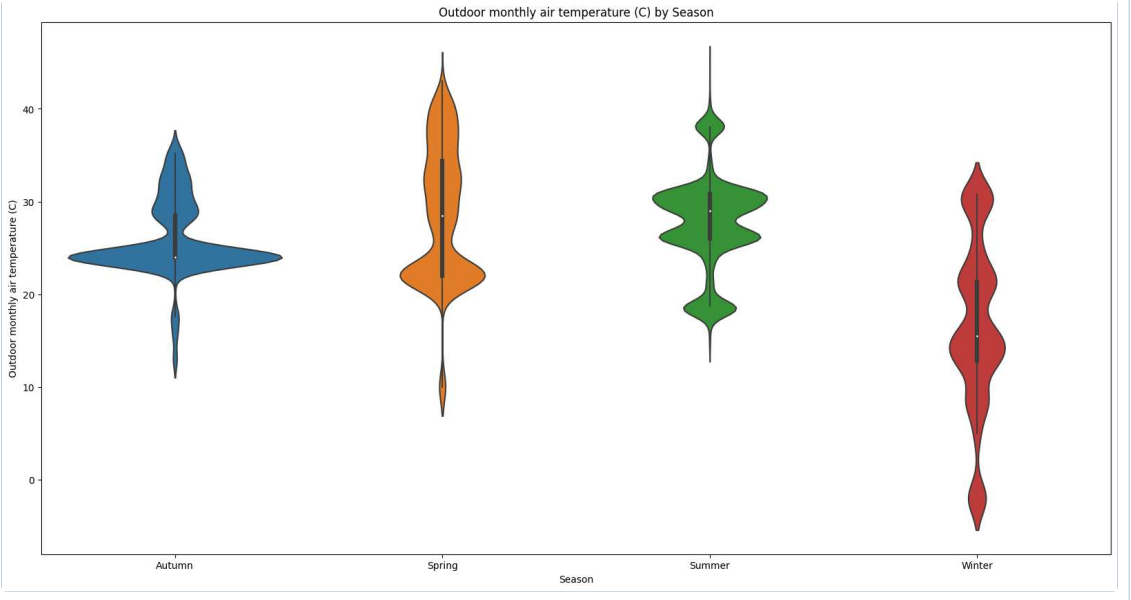
23

```
plt.figure(figsize=(20, 10))
sns.boxplot(x='Season', y='Air temperature (C)', data=data_df)
plt.xlabel('Season')
plt.ylabel('Air Temperature (C)')
plt.title('Air Temperature by Season')
plt.show()
```

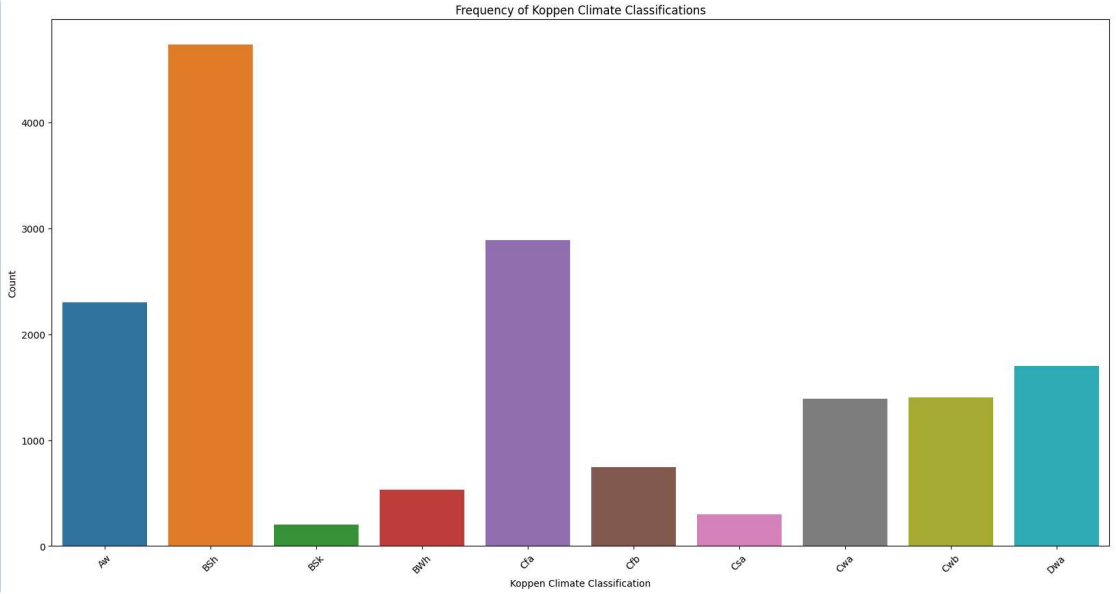


Violin Plots Density of the data distribution.


```
plt.figure(figsize=(20, 10))
sns.violinplot(x='Season', y='Outdoor monthly air temperature (C)', data=data_df)
plt.xlabel('Season')
plt.ylabel('Outdoor monthly air temperature (C)')
plt.title('Outdoor monthly air temperature (C) by Season')
plt.show()
```



```
plt.figure(figsize=(20, 10))
sns.countplot(x='Koppen climate classification', data=data_df)
plt.xlabel('Koppen Climate Classification')
plt.ylabel('Count')
plt.title('Frequency of Koppen Climate Classifications')
plt.xticks(rotation=45)  # Rotate x-axis labels if needed
plt.show()
```



6. Model Training


```
Y = data_df['Thermal comfort']

features = ['Air temperature (C)', 'Air velocity (m/s)', 'Relative humidity (%)', 'Clo', 'Met',
            'Season', 'Koppen climate classification', 'Building type', 'Cooling startegy_building level',
'Outdoor monthly air temperature (C)']

X = data_df[features]

X.describe()
```

	Air temperature (C)	Air velocity (m/s)	Relative humidity (%)	Clo	Outdoor monthly air temperature (C)
count	16209.000000	16209.000000	16209.000000	16209.000000	16209.000000
mean	25.388728	0.261687	50.913968	0.681518	24.768153
std	4.439810	0.693904	15.763509	0.324204	8.259837
min	0.600000	0.000000	10.300000	0.000000	-2.000000
25%	22.800000	0.060000	39.400000	0.490000	21.300000
50%	25.300000	0.150000	50.100000	0.630000	26.000000
75%	28.100000	0.330000	63.300000	0.780000	30.800000
max	45.200000	56.170000	95.300000	2.870000	45.100000

```
Y.describe()
```

```
count     16209
unique       17
top           5
freq       6230
Name: Thermal comfort, dtype: object
```

30

```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16209 entries, 74 to 81750
Data columns (total 11 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   Air temperature (C)                 16209 non-null  float64
 1   Air velocity (m/s)                  16209 non-null  float64
 2   Relative humidity (%)               16209 non-null  float64
 3   Clo                                 16209 non-null  float64
 4   Met                                 16209 non-null  category
 5   Season                              16209 non-null  category
 6   Koppen climate classification       16209 non-null  category
 7   Building type                       16209 non-null  category
 8   Cooling startegy_building level     16209 non-null  category
 9   Outdoor monthly air temperature (C) 16209 non-null  float64
 10  Thermal comfort                     16209 non-null  category
dtypes: category(6), float64(5)
memory usage: 857.1 KB
```

31

```
print("Shape of X before splitting:", X.shape)
print("Shape of Y before splitting:", Y.shape)
```

```
Shape of X before splitting: (16209, 10)
Shape of Y before splitting: (16209,)
```

32

```
# One-hot encode categorical features
categorical_features = ['Met','Season', 'Koppen climate classification', 'Building type', 'Cooling
startegy_building level']
encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
encoded_features = encoder.fit_transform(X[categorical_features])
encoded_df = pd.DataFrame(encoded_features)
```

```
/databricks/python/lib/python3.11/site-packages/sklearn/preprocessing/_encoders.py:972: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `s
parse` to its default value.
  warnings.warn(
```

33

```
X = X.drop(categorical_features, axis=1)
X_np = X.values
encoded_np = encoded_df.values

# Horizontally stack the arrays
X_combined = np.hstack((X_np, encoded_np))

# Convert back to a pandas DataFrame (optional)
X = pd.DataFrame(X_combined)
```

34

```
print("Shape of X before splitting:", X.shape)
print("Shape of Y before splitting:", Y.shape)
```

```
Shape of X before splitting: (16209, 51)
Shape of Y before splitting: (16209,)
```

35

```
# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Initialize and train the model
#model = RandomForestRegressor(random_state=42)
model = RandomForestClassifier(random_state=42)

model.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred = model.predict(X_test)

# Evaluate the model
#mse = mean_squared_error(Y_test, Y_pred)
#print(f'Mean Squared Error: {mse}')
# Calculate MAE
# Calculate RMSE manually
#rmse = np.sqrt(mse)

# Calculate MAE
#mae = mean_absolute_error(Y_test, Y_pred)

# Calculate R-squared
#r2 = r2_score(Y_test, Y_pred)

#print(f'RMSE: {rmse}')
#print(f'MAE: {mae}')
#print(f'R-squared: {r2}')
```

7. Evaluation Matrix

37

```
of this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/databricks/python/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/databricks/python/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/databricks/python/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/databricks/python/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```