Data Analysis and Machine-Learning

Chapter 10.1.

ML Modelling Applications (1)



*Knowledge without practice is useless.*

*Practice without knowledge is dangerous.*

*- Confucius*

1. Introduction

So far we have covered the algorithmic concepts and mathematical calculations regarding various ML models, including Linear Models, Generalized Linear Models, Support Vector Machine, K-Nearest Neighbor, Decision-Tree Model, Random Forest Model, etc. We have also covered various data-science techniques for effective analytics and modelling, including feature selection methods (FS, BE, SS), Multicollinearity/Variance Inflation Factor processing, Variable Weight Evaluation Methodologies (e.g., eli5, Permutation Importance, Shap), scaling, stochastic modelling, regularizations (L1, L2), etc.

In other words, we are now equipped with the necessary skillsets to perform actual predictions using data. In actual applications, EDA (Exploratory Data Analysis), descriptive statistics of variables, visualizations, and preprocessing consist a big part in prior to the actual modelling process, as deeply learning about the data per se is essential to generate an effective model with best accuracy. Focusing on such aspects together with the purpose of demonstrating various modelling methodologies, this chapter will be focusing on various applications of ML analysis & models via data coding. In this subchapter in particular, we will be predicting the future bicycle rental volumes (counts) using existing data on various meteorological measures.

2. EDA (Exploratory Data Analysis) and Visualizations of Variables

#Import Essential Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sys

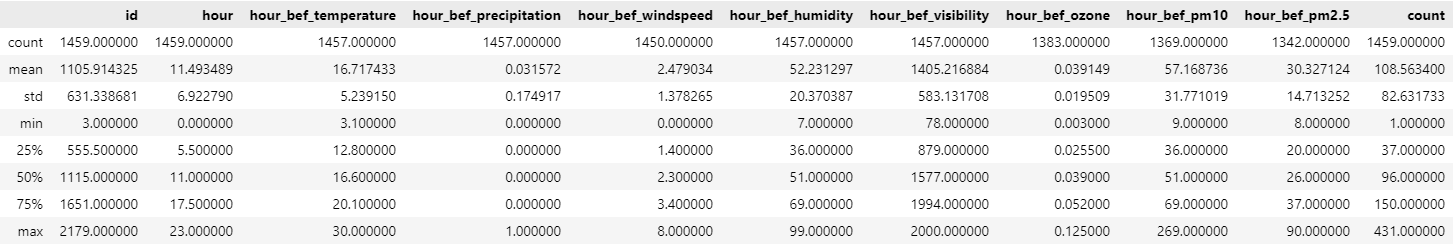
sys.path.append('C:\\Users\\Master\\Desktop\\dataTools')

import dataTools as dt

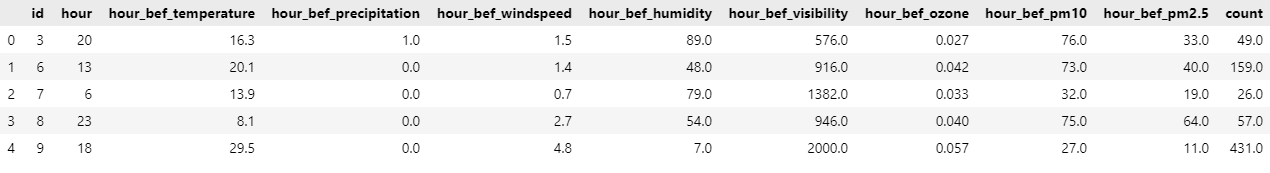
train = pd.read\_csv('Your File Path\\train.csv')

test = pd.read\_csv('Your File Path\\test.csv')

train.describe()



train.head()



display(len(train.columns))

display(len(test.columns))

#Test data has one less column (Count, which is our prediction target)

Output:

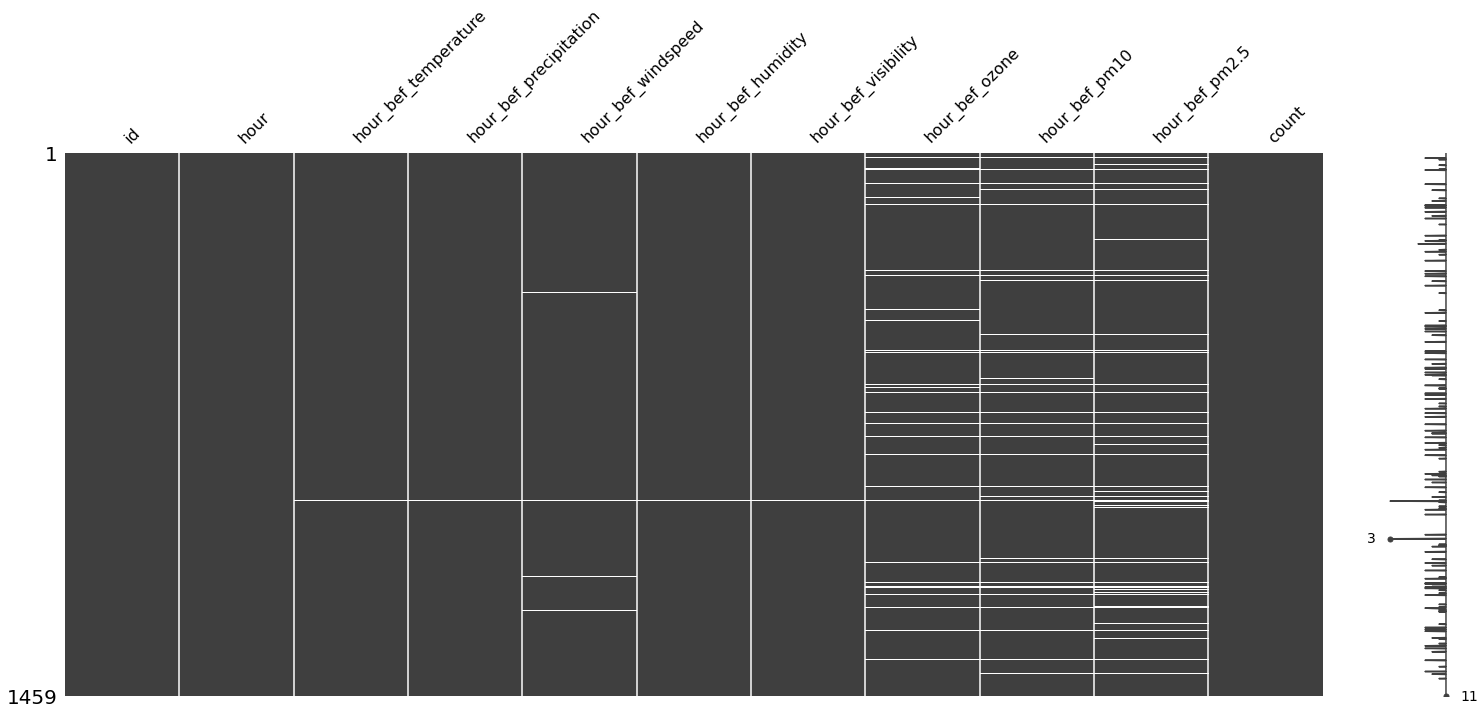
11

10

#Missing Values

import missingno

missingno.matrix(train)

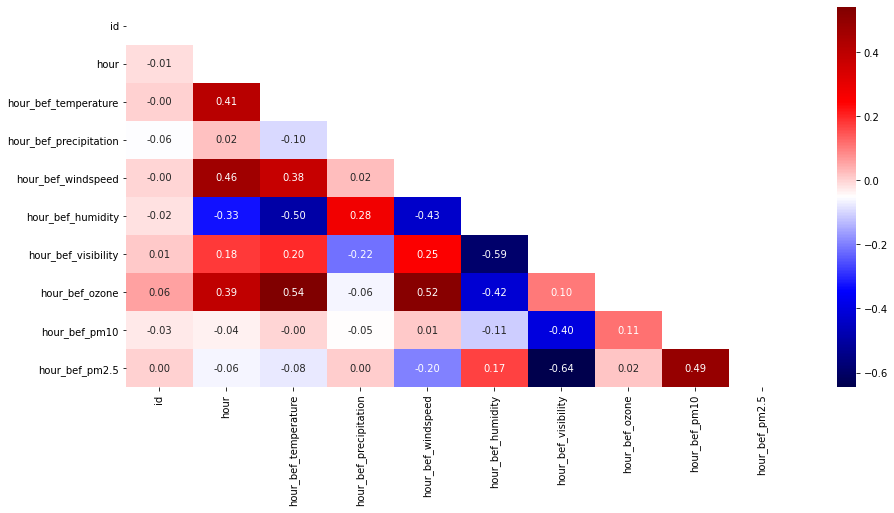


White horizontal lines display missing values. Notice that pm2.5, pm10, ozone data have relatively more missing values compared to other data. Missing values should be pre-processed in prior to the actual modelling process.

2.1. X Variables: Data Shapes and Relationships

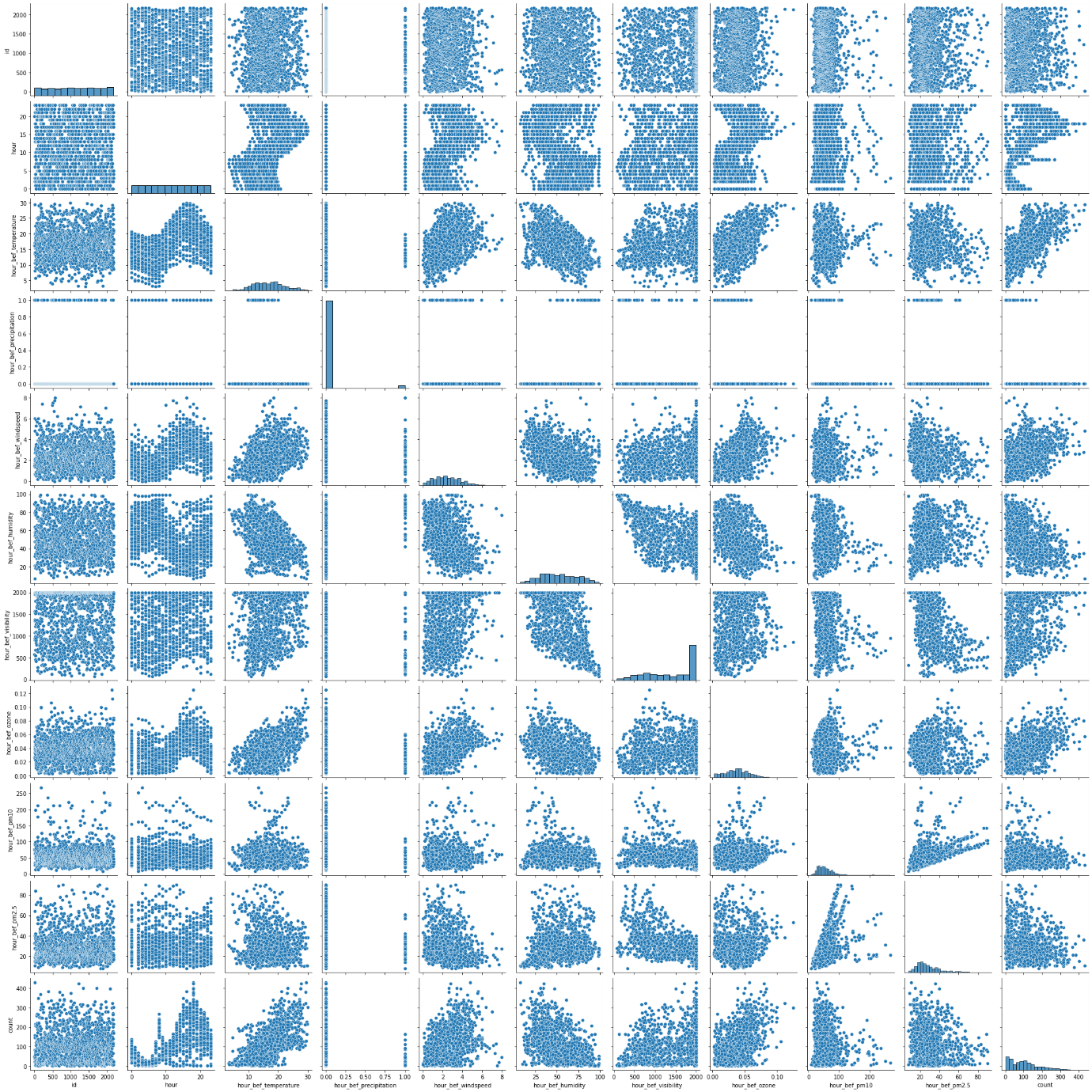
#Correlations (among Xs)

dt.visualCorr(train.drop(columns='count'))



#Pairplot

sns.pairplot(data=train)

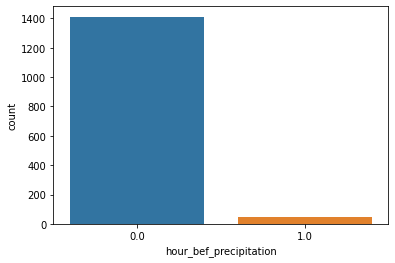


#Spotting Categorical Variables

train['hour\_bef\_precipitation'].value\_counts()

#Simple Visualization

sns.countplot(data=train, x='hour\_bef\_precipitation')



train['hour\_bef\_temperature'].unique()

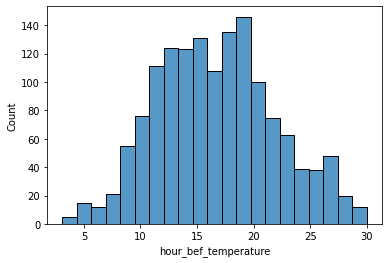
array([16.3, 20.1, 13.9, 8.1, 29.5, 13.6, 10.6, 16. , 13.8, 17.2, 5.7, 15.4, 14.1, 9.2, 20. , 14. , 18.8, 11.5, 22.6, 18. , 12.6, 19.4, 11.7, 13. , 10.9, 8.9, 16.8, 20.8, 11.4, 10.2, 23.4, 21.2, 19.8, 19.7, 15.3, 18.9, 14.9, 20.2, 15.1, 21.7, 25.1, 12.2, 14.6, 11. , 15.6, 16.6, 14.7, 12.1, 13.5, 15. , 18.6, 22.3, 12.5, 25.3, 21.1, 12.7, 17. , 24.7, 18.1, 6.4, 20.4, 16.2, 15.5, 22.1, 29.1, 13.2, 9.4, 17.7, 10.7, 19.9, 20.7, 22. , 8.2, 17.8, 25. , 14.3, 23.3, 15.8, 21.5, 8. , 8.5, 25.4, 10.1, 29.6, 16.9, 18.3, 21.8, 9.5, 28.4, 22.7, 13.3, 12.9, 26.2, 18.5, 27.2, 27. , 18.7, 10.4, 8.4, 19.3, 22.2, 18.2, 15.2, 10.5, 23.6, 17.6, 18.4, 26.4, 25.7, 17.4, 23.2, 7.5, 12.4, 12.3, 11.6, 14.2, 9.9, 8.8, 11.1, 13.4, 19.6, 21.4, 7. , 7.9, 23.5, 16.4, 13.1, 21.3, 23. , 27.5, 6.6, 22.8, 10.3, 14.4, 15.9, 11.8, 23.1, 19.2, 8.7, 24. , 16.1, 23.7, 11.2, 17.3, 19.5, 26.6, 19. , 22.5, 24.1, 7.7, 14.5, 25.6, 19.1, 4.4, 29. , 16.5, 28.8, 11.9, 10.8, 26.8, 9.1, 13.7, 4. , 10. , 28.7, 9. , 14.8, 26.3, 20.9, 28.1, 12.8, 5. , 9.8, 27.3, 21.9, 20.6, 24.2, 26.5, 24.4, 20.5, 15.7, 20.3, 21.6, 17.9, 29.3, 9.7, 22.9, 7.2, 24.6, 5.3, 4.6, 4.9, 17.1, 21. , 25.9, 5.6, 11.3, 24.8, 26.1, 27.1, 27.7, 23.9, 17.5, 24.3, 23.8, 26.7, 30. , 16.7, 12. , 9.3, 5.5, 6. , 22.4, 27.4, 7.1, 8.6, 3.2, 27.8, 7.6, 25.5, 24.5, 6.3, 5.4, 3.1, 4.2, nan, 29.8, 5.9, 6.8, 9.6, 29.4, 26. , 6.7, 25.8, 4.5, 27.9, 25.2, 26.9, 8.3, 7.3, 5.1, 3.3, 28. , 7.4, 27.6, 29.2])

train['hour\_bef\_temperature'].value\_counts()

18.8 17 19.4 17 18.0 16 14.0 16 16.6 15 .. 4.0 1 29.0 1 4.6 1 26.5 1 29.5 1 Name: hour\_bef\_temperature, Length: 245, dtype: int64

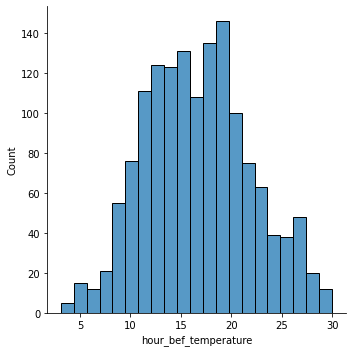
#Histplot

sns.histplot(data=train, x='hour\_bef\_temperature')



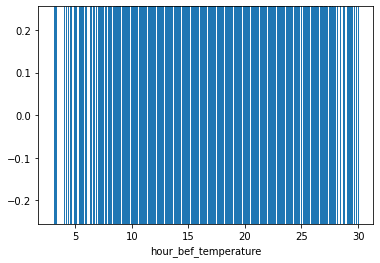
#Displot

sns.displot(data=train, x='hour\_bef\_temperature')



#Rugplot

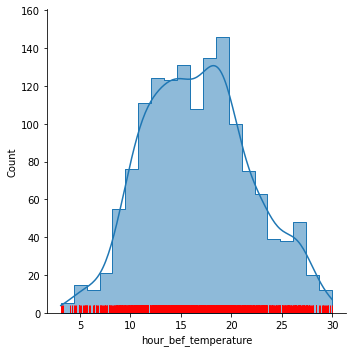
sns.rugplot(data=train, x='hour\_bef\_temperature', height=1)



#Displot with trend lines and rugplot

sns.displot(data=train, x='hour\_bef\_temperature', rug=True, rug\_kws={'color':'r'}, kde=True, element='step')

plt.show()



train.columns

Index(['id', 'hour', 'hour\_bef\_temperature', 'hour\_bef\_precipitation', 'hour\_bef\_windspeed', 'hour\_bef\_humidity', 'hour\_bef\_visibility', 'hour\_bef\_ozone', 'hour\_bef\_pm10', 'hour\_bef\_pm2.5', 'count'], dtype='object')

train['hour\_bef\_windspeed'].unique()

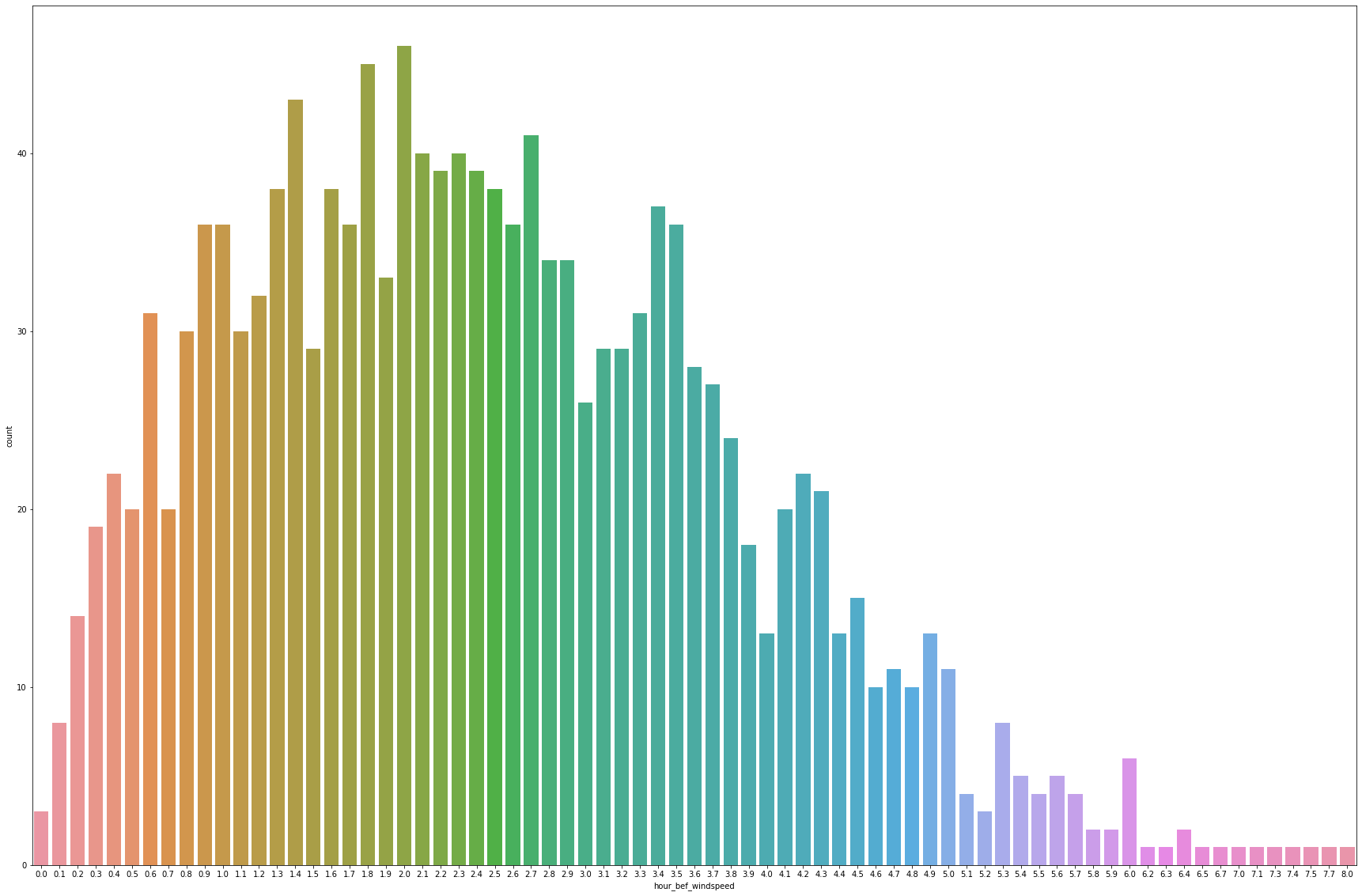
array([1.5, 1.4, 0.7, 2.7, 4.8, 1.7, 6. , 1.9, 2.1, 0.6, 3.2, 1.8, 2.8, 2.2, 3. , nan, 3.1, 5.3, 5.9, 1.6, 3.5, 0.5, 0.8, 3.3, 0.9, 4.6, 2.6, 2.3, 4.5, 2.5, 2.9, 5.7, 1.3, 1.2, 0.2, 1. , 3.9, 2. , 2.4, 4.9, 4.1, 5.5, 0.4, 3.7, 3.6, 1.1, 3.4, 3.8, 4. , 5.1, 4.3, 4.4, 7.4, 0.3, 4.7, 4.2, 5.8, 5. , 5.6, 5.4, 7.3, 7.5, 0.1, 6.7, 0. , 7.7, 8. , 6.2, 5.2, 6.4, 7.1, 6.5, 6.3, 7. ])

train['hour\_bef\_windspeed'].value\_counts()

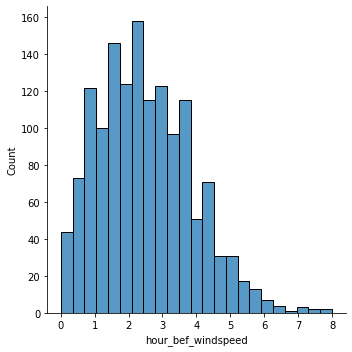
2.0 46 1.8 45 1.4 43 2.7 41 2.3 40 .. 7.7 1 6.3 1 7.4 1 6.7 1 7.3 1 Name: hour\_bef\_windspeed, Length: 73, dtype: int64

plt.figure(figsize=(30,20))

sns.countplot(data=train, x='hour\_bef\_windspeed')



sns.displot(data=train, x='hour\_bef\_windspeed')

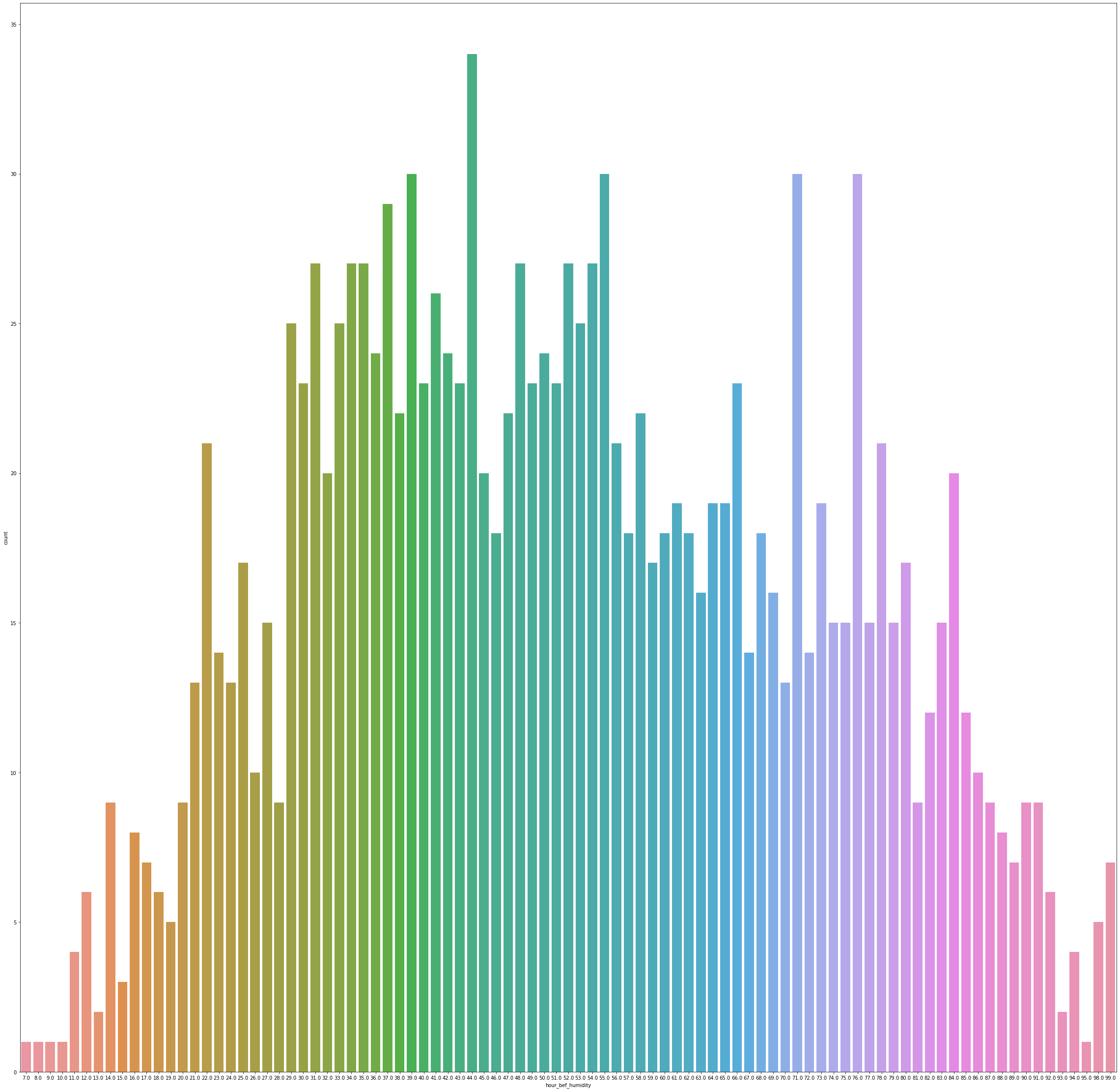


train['hour\_bef\_humidity'].value\_counts()

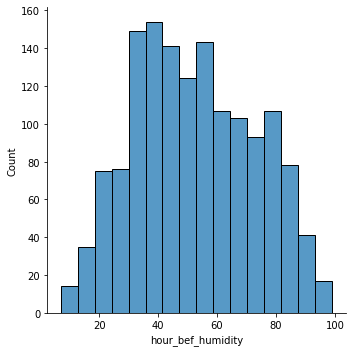
44.0 34 71.0 30 55.0 30 76.0 30 39.0 30 .. 10.0 1 7.0 1 8.0 1 9.0 1 95.0 1 Name: hour\_bef\_humidity, Length: 91, dtype: int64

plt.figure(figsize=(40,40))

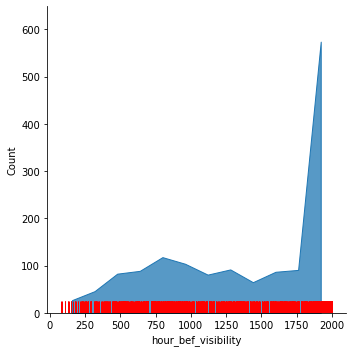
sns.countplot(data=train, x='hour\_bef\_humidity')



sns.displot(data=train, x='hour\_bef\_humidity')



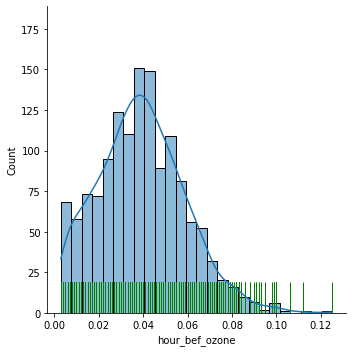
sns.displot(data=train, x='hour\_bef\_visibility', rug=True, rug\_kws={'color':'red', 'height':0.04}, element='poly')



train['hour\_bef\_ozone'].value\_counts()

0.036 34 0.040 33 0.044 32 0.042 32 0.033 31 .. 0.125 1 0.093 1 0.112 1 0.090 1 0.106 1 Name: hour\_bef\_ozone, Length: 95, dtype: int64

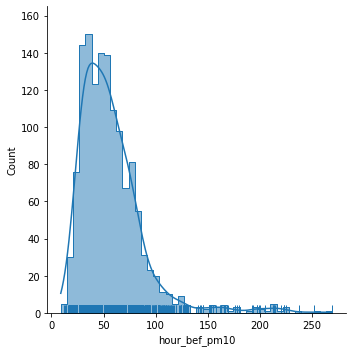
sns.displot(data=train, x='hour\_bef\_ozone', rug=True, rug\_kws={'height':0.1, 'color':'g'}, kde=True)



train['hour\_bef\_pm10'].value\_counts()

32.0 36 34.0 33 48.0 31 54.0 28 47.0 28 .. 127.0 1 131.0 1 111.0 1 89.0 1 129.0 1 Name: hour\_bef\_pm10, Length: 148, dtype: int64

sns.displot(data=train, x='hour\_bef\_pm10', kde=True, rug=True, element='step')

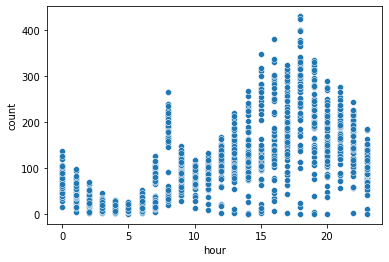


2.2. Relationship between X Variables and Y (Target)

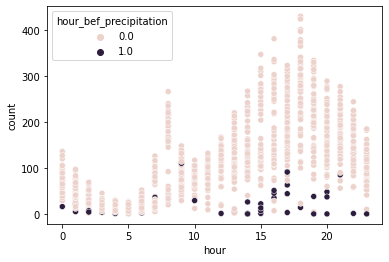
Relationship between X and Y can be effectively displayed using scatterplots, relationship plots (relplot), and box plots.

#Relationship between Time and Rental Counts

sns.scatterplot(data=train, x='hour', y='count')

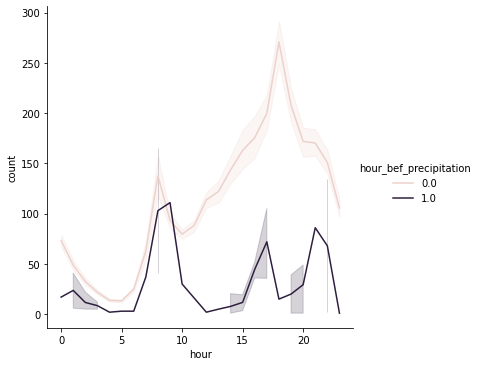


sns.scatterplot(data=train, x='hour', y='count', hue='hour\_bef\_precipitation')



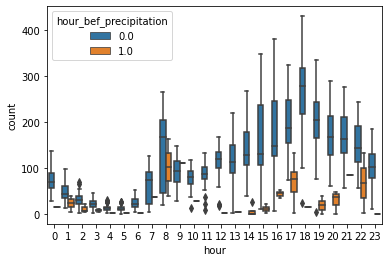
Notice that the rental rates of bicycles are especially high around 8:00~9:00 and 16:00~20:00, which makes sense considering typical commuting times. Also, notice that the rental rates are consistently low when ‘hour\_bef\_precipitation’ value is 1.0, which means that rental rates are low during rainfalls. Highlighting more on the part of precipitation, the same plot can also be displayed as:

sns.relplot(data=train, x='hour', y='count', kind='line', hue='hour\_bef\_precipitation')



Or, using box plots,

sns.boxplot(data=train, x='hour', y='count', hue='hour\_bef\_precipitation')



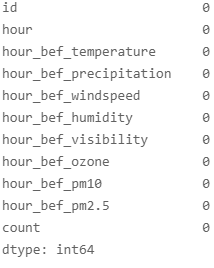
Notice that the data dispersion ranges of counts are generally large, especially after 13:00, while there are relatively little data considered as outliers.

3. Demonstration of Preprocessing

#Processing Missing Values (Elimination)

train.dropna(axis='index', inplace=True)

train.isna().sum()



#Processing Outliers (for demonstration purpose)

display(train['count'].quantile(0.99))

display(train['count'].quantile(0.01))

display(train['count'].mean())

display(train['count'].median())

display(train['count'].std())

high = train['count'].quantile(0.99)

low = train['count'].quantile(0.01)

train = train[(high > train['count']) & (low < train['count'])]

#Scaling for Continuous Data

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df\_cont = train[[

    'hour\_bef\_temperature',

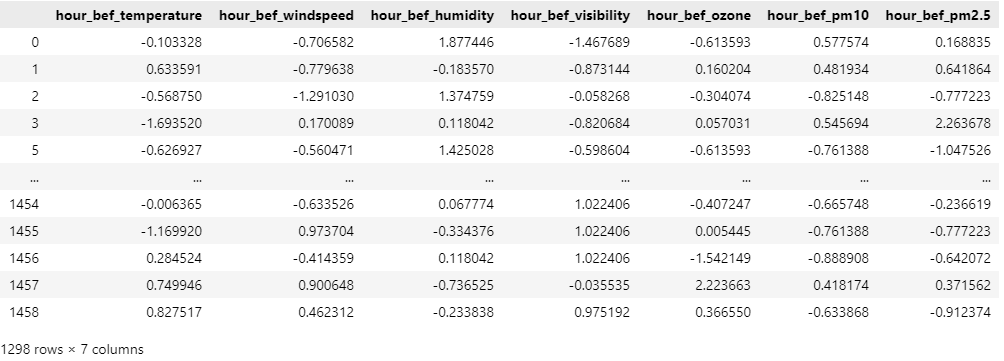
    'hour\_bef\_windspeed', 'hour\_bef\_humidity', 'hour\_bef\_visibility',

    'hour\_bef\_ozone', 'hour\_bef\_pm10', 'hour\_bef\_pm2.5'

]]

df\_cont = pd.DataFrame(scaler.fit\_transform(X=df\_cont), columns=df\_cont.columns, index=df\_cont.index)

df\_cont



#Processing Categorical Data: Using Dummy Variables

train['hour\_bef\_precipitation'].value\_counts()

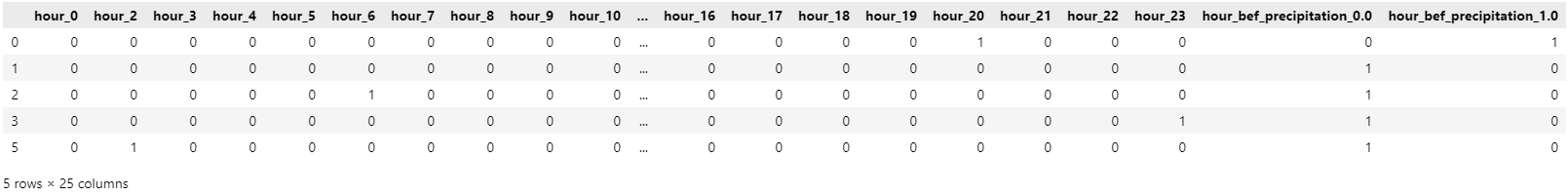
df\_cat = train[[

    'hour', 'hour\_bef\_precipitation'

]]

df\_cat = pd.get\_dummies(data=df\_cat, columns=df\_cat.columns)

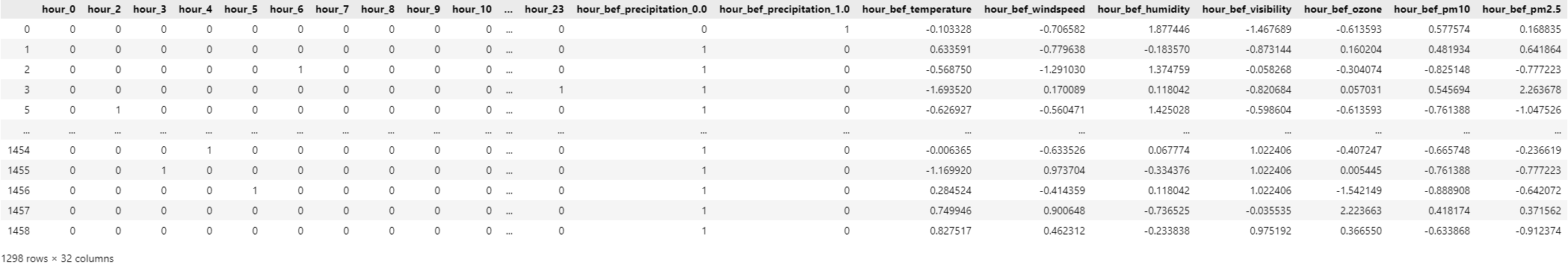
df\_cat.head()



#Merge Continuous (scaled) data and Categorical (Dummies) data

x = pd.concat([df\_cat, df\_cont], axis=1)

x



y = train['count']

4. Modelling

4.1. Base Model

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

#Train-Test Split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, shuffle=True, random\_state=123)

x\_train = x\_train.sort\_index()

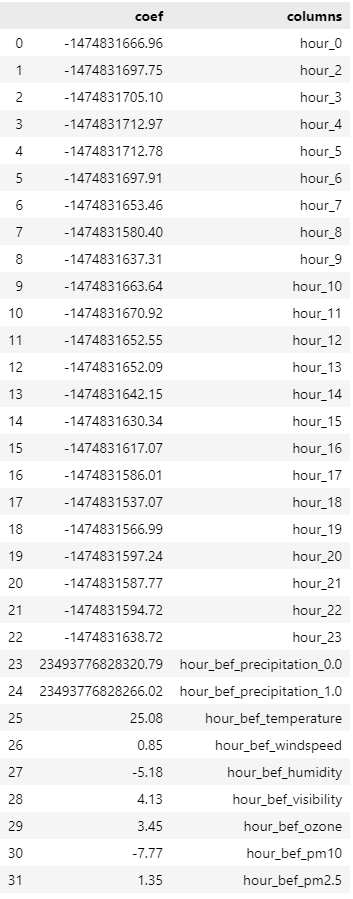
y\_train = y\_train.sort\_index()

x\_test = x\_test.sort\_index()

y\_test = y\_test.sort\_index()

baseModel = LinearRegression().fit(x\_train, y\_train)

dt.coef(baseModel, x\_train)



Notice that the coefficients of the dummy variables are overwhelmingly large compared to the coefficients of the rest of the variables, which may cause distortion of the result. Thus, it would be better to perform regularization (L2, L1) of the coefficients, as follows.

#Regularization

from sklearn.linear\_model import ElasticNet

from sklearn.model\_selection import GridSearchCV, KFold

folds = KFold(n\_splits=10, shuffle=True)

elasticModel = ElasticNet()

#Gridsearch

params = {

    'l1\_ratio': [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],

    'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

}

search = GridSearchCV(elasticModel, param\_grid=params, scoring='r2', cv=folds)

search.fit(x\_train, y\_train)

GridSearchCV(cv=KFold(n\_splits=10, random\_state=None, shuffle=True), estimator=ElasticNet(), param\_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000], 'l1\_ratio': [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]}, scoring='r2')

display(search.best\_params\_)

display(search.best\_score\_)

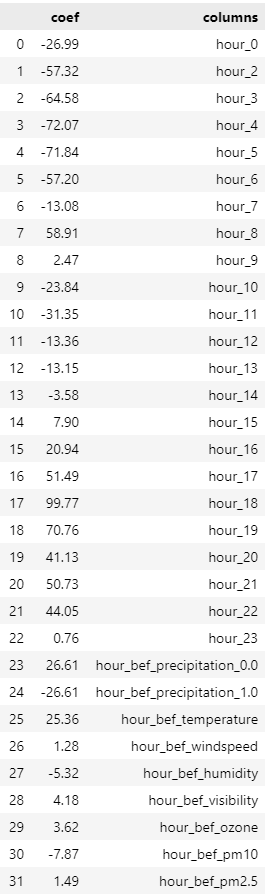
display(search.best\_estimator\_)

{'alpha': 0.001, 'l1\_ratio': 0.5}

0.7269735690900276

ElasticNet(alpha=0.001)

dt.coef(search.best\_estimator\_, x\_train)



elasticModel = ElasticNet(alpha=0.001).fit(x\_train, y\_train)

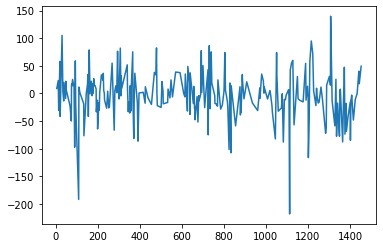
yhat = elasticModel.predict(x\_test)

#How are the residuals moving?

resid = y\_test - yhat

plt.plot(resid)

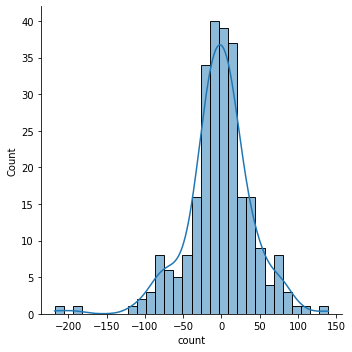
plt.show()



#Distribution of the Residuals (Supposition of Normal Distribution)

resid = y\_test-yhat

sns.displot(x=resid, kde=True)



dt.rSquare(x\_test, y\_test, yhat)

{'r2': 0.71359171971959, 'adjr2': 0.6732169841734529}

4.2. Support Vector Machine

from sklearn.svm import SVR

#Cross-Validation

folds = KFold(n\_splits=10, shuffle=True)

#Gridsearch

svmModel = SVR()

params = {

    'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

    'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

}

search = GridSearchCV(svmModel, param\_grid=params, scoring='r2', cv=folds)

search.fit(x\_train, y\_train)

GridSearchCV(cv=KFold(n\_splits=10, random\_state=None, shuffle=True), estimator=SVR(), param\_grid={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000], 'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}, scoring='r2')

search.best\_params\_

{'C': 100, 'gamma': 0.1}

search.best\_estimator\_

SVR(C=100, gamma=0.1)

svmModel\_best = SVR(C=100, gamma=0.1).fit(x\_train, y\_train)

yhat = svmModel\_best.predict(x\_test)

dt.rSquare(x\_test, y\_test, yhat)

{'r2': 0.7312528179519671, 'adjr2': 0.6933677526412312}

#Residuals

resid = y\_test - yhat

plt.figure(figsize=(10,5))

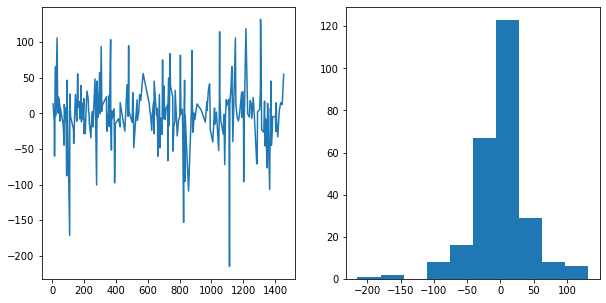
plt.subplot(1,2,1)

plt.plot(resid)

plt.subplot(1,2,2)

plt.hist(resid)

plt.show()



4.3. Random Forest Model (Base Model)

#Random Forest (Base)

from sklearn.ensemble import RandomForestRegressor

base\_rfModel = RandomForestRegressor().fit(x\_train, y\_train)

yhat = base\_rfModel.predict(x\_test)

dt.rSquare(x\_test, y\_test, yhat)

{'r2': 0.7261433026249263, 'adjr2': 0.6875379532152243}

#Feature Importance

importance = base\_rfModel.feature\_importances\_

importance

array([0.00426548, 0.01044324, 0.01091615, 0.0096999 , 0.00860604, 0.00812217, 0.00277887, 0.03823114, 0.003626 , 0.0030072 , 0.00318027, 0.00213123, 0.00303486, 0.00395847, 0.00362613, 0.00285532, 0.00438218, 0.03654878, 0.01722575, 0.00606158, 0.01312431, 0.01824675, 0.00475323, 0.00087787, 0.00152762, 0.39858978, 0.11836999, 0.04712191, 0.06344056, 0.07048591, 0.04555722, 0.03520408])

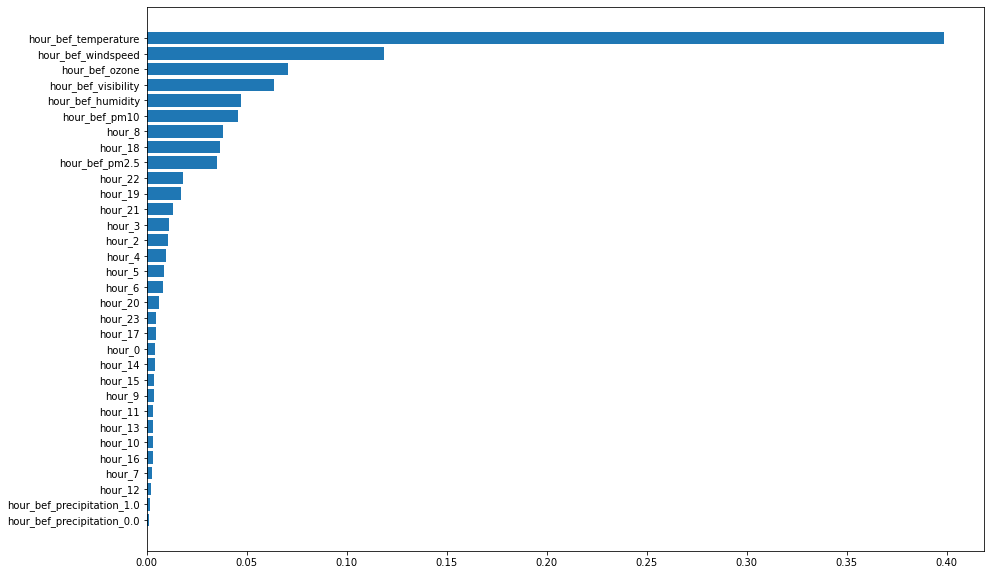
np.argsort(importance)

array([23, 24, 11, 6, 15, 9, 12, 10, 8, 14, 13, 0, 16, 22, 19, 5, 4, 3, 1, 2, 20, 18, 21, 31, 17, 7, 30, 27, 28, 29, 26, 25], dtype=int64)

indice = np.argsort(importance)

plt.figure(figsize=(15,10))

plt.barh(y= x\_train.columns[indice], width=importance[indice], align='center')



4.4. LightGBM

import lightgbm as lgb

lgbModel = lgb.LGBMRegressor().fit(x\_train, y\_train)

yhat = lgbModel.predict(x\_test)

dt.rSquare(x\_test, y\_test, yhat)

{'r2': 0.7374491848024447, 'adjr2': 0.7004376161402343}

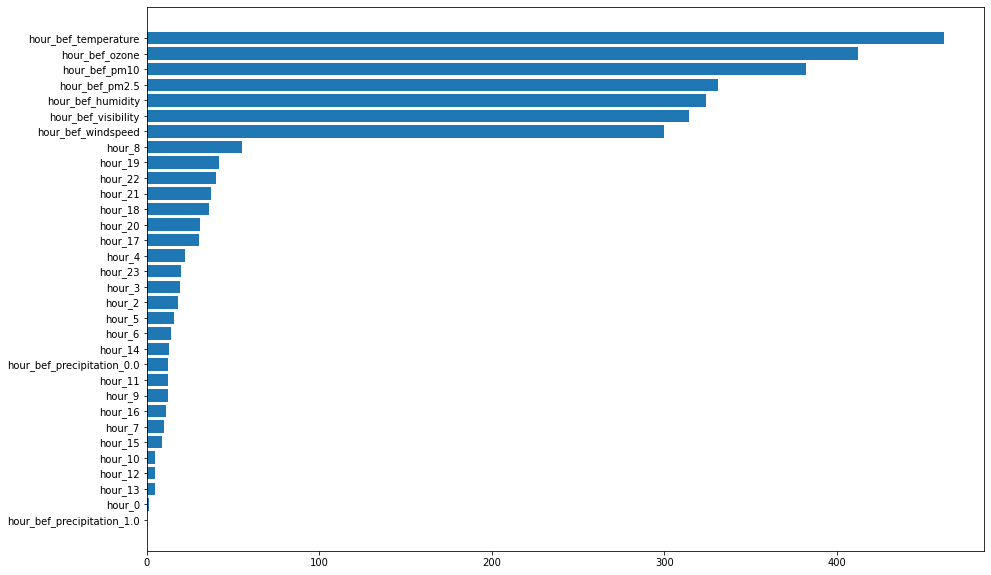
#Feature Importance

importance = lgbModel.feature\_importances\_

indice = np.argsort(importance)

plt.figure(figsize=(15,10))

plt.barh(y = x\_train.columns[indice], width=importance[indice], align='center')



4.5. Random Forest Model (Optimization)

rfModel = RandomForestRegressor()

params = {

    'n\_estimators': [100, 300, 500],

    'max\_depth': [3, 5, 7, None],

    'criterion': ['mse', 'mae'],

    'max\_features': ['auto', 'sqrt', 'log2'],

    'min\_samples\_split':[3, 5, 7],

    'min\_samples\_leaf':[1, 3, 7],

}

search = GridSearchCV(rfModel, param\_grid=params, cv=folds, scoring='r2')

search.fit(x\_train, y\_train)

GridSearchCV(cv=KFold(n\_splits=10, random\_state=None, shuffle=True), estimator=RandomForestRegressor(), param\_grid={'criterion': ['mse', 'mae'], 'max\_depth': [3, 5, 7, None], 'max\_features': ['auto', 'sqrt', 'log2'], 'min\_samples\_leaf': [1, 3, 7], 'min\_samples\_split': [3, 5, 7], 'n\_estimators': [100, 300, 500]}, scoring='r2')

display(search.best\_params\_)

display(search.best\_score\_)

display(search.best\_estimator\_)

{'criterion': 'mae', 'max\_depth': None, 'max\_features': 'sqrt', 'min\_samples\_leaf': 1, 'min\_samples\_split': 3, 'n\_estimators': 500}

0.7292339867822705

RandomForestRegressor(criterion='mae', max\_features='sqrt', min\_samples\_split=3, n\_estimators=500)

best\_rfModel = RandomForestRegressor(criterion='mae', max\_features='sqrt', min\_samples\_split=3, n\_estimators=500).fit(x\_train, y\_train)

yhat = best\_rfModel.predict(x\_test)

dt.rSquare(x\_test, y\_test, yhat)

{'r2': 0.7518152541991256, 'adjr2': 0.716828858315302}

importance = best\_rfModel.feature\_importances\_

importance

array([0.00780525, 0.02210177, 0.03258889, 0.02959408, 0.02577432, 0.02165343, 0.00458194, 0.02388403, 0.00568792, 0.00705419, 0.00725494, 0.00437371, 0.00556425, 0.00499584, 0.00514896, 0.00558884, 0.01027409, 0.02980541, 0.02379161, 0.01216738, 0.0172746 , 0.01600201, 0.0060864 , 0.0033544 , 0.00330947, 0.16975497, 0.10246193, 0.09678073, 0.06957318, 0.10651645, 0.06334513, 0.05584988])

indice = np.argsort(importance)

plt.figure(figsize=(15,10))

plt.barh(y = x\_train.columns[indice], width=importance[indice], align='center')

