```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("D:\\DATA ANALYST PYTHON\\Projects\\Bike Sharing
Analysis\\1584513771_bikesharingdemandanalysis\\hour.csv")

df.head()
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	\
0	1	2011-01-01	1	0	1	0	0	6	0	
1	2	2011-01-01	1	0	1	1	0	6	0	
2	3	2011-01-01	1	0	1	2	0	6	0	
3	4	2011-01-01	1	0	1	3	0	6	0	
4	5	2011-01-01	1	0	1	4	0	6	0	

	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	0.24	0.2879	0.81	0.0	3	13	16
1	1	0.22	0.2727	0.80	0.0	8	32	40
2	1	0.22	0.2727	0.80	0.0	5	27	32
3	1	0.24	0.2879	0.75	0.0	3	10	13
4	1	0.24	0.2879	0.75	0.0	0	1	1

```
# Check for null values in the data and drop records with NAs.

df.isnull().sum()
```

```
instant       0
dteday        0
season        0
yr            0
mnth          0
hr            0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
```

```
atemp          0
hum            0
windspeed      0
casual         0
registered     0
cnt            0
dtype: int64
```

```
# Sanity checks:

# Check if registered + casual = cnt for all the records. If not, the
row is junk and should be dropped.

# Month values should be 1-12 only

# Hour values should be 0-23
```

```
df['registered']+df['casual'] == df['cnt']
```

```
0         True
1         True
2         True
3         True
4         True
          ...
17374     True
17375     True
17376     True
17377     True
17378     True
Length: 17379, dtype: bool
```

```
np.unique(df['mnth'])
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12], dtype=int64)
```

```
np.unique(df['hr'])
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16,
       17, 18, 19, 20, 21, 22, 23], dtype=int64)
```

```
# The variables ‘casual’ and ‘registered’ are redundant and need to be
dropped. ‘Instant’ is the index and needs to be dropped too. The date
column dteday will not be used in the model building, and therefore
needs to be dropped. Create a new dataframe named inp1.
```

```
data = ['casual','registered','instant','dteday']
```

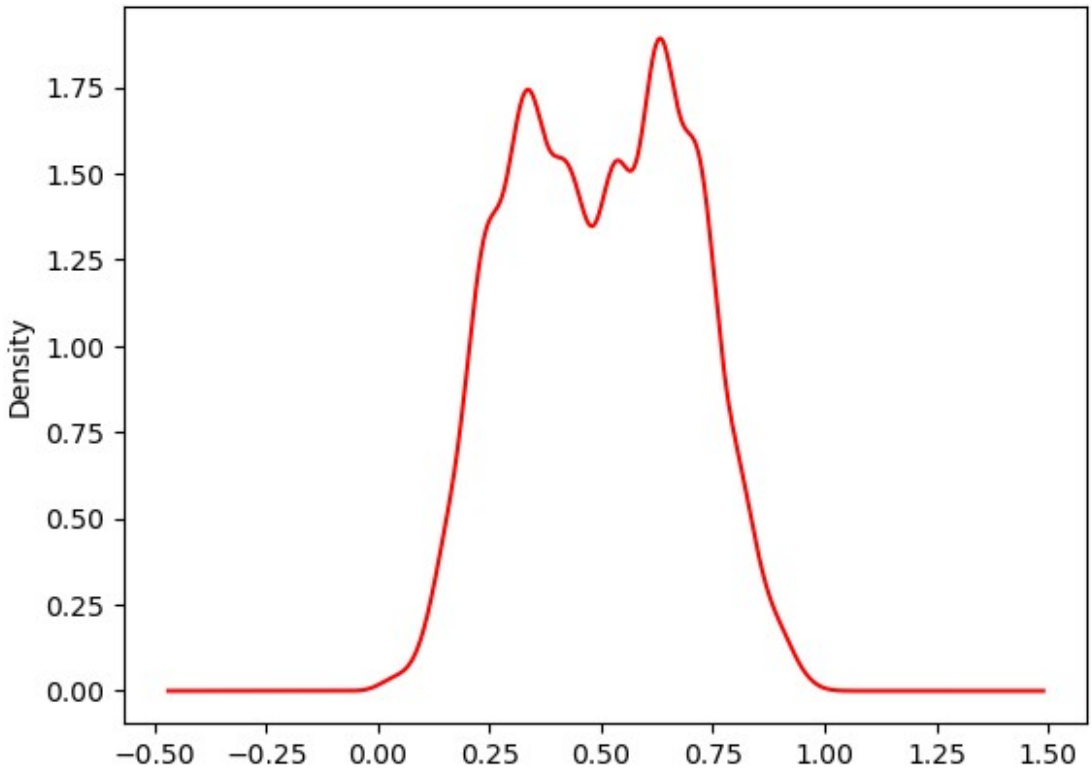
```
inp1 = df.drop(data,axis=1).copy()
```

```
# Univariate analysis:

# Describe the numerical fields in the dataset using pandas describe
method.

df.describe()
```

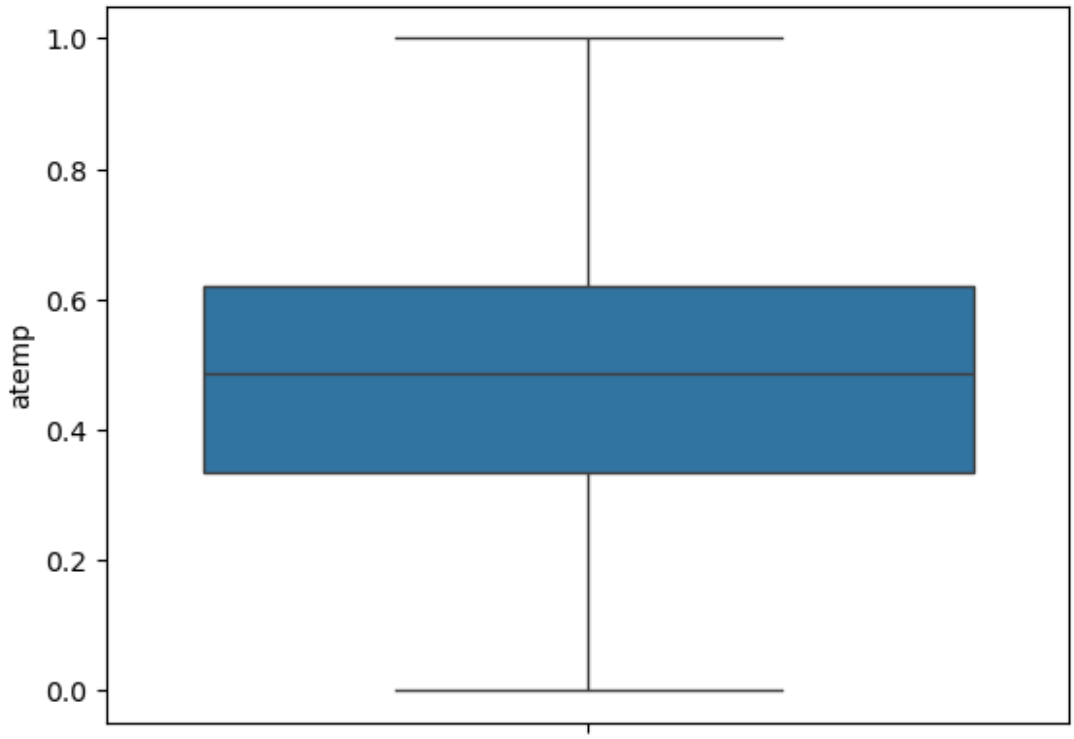
```
          instant        season            yr          mnth
hr  \
count  17379.0000  17379.000000  17379.000000  17379.000000
17379.000000
mean    8690.0000      2.501640      0.502561      6.537775
11.546752
std     5017.0295      1.106918      0.500008      3.438776
6.914405
min        1.0000      1.000000      0.000000      1.000000
0.000000
25%     4345.5000      2.000000      0.000000      4.000000
6.000000
50%     8690.0000      3.000000      1.000000      7.000000
12.000000
75%    13034.5000      3.000000      1.000000     10.000000
18.000000
max    17379.0000      4.000000      1.000000     12.000000
23.000000

            holiday       weekday    workingday    weathersit
temp  \
count  17379.000000  17379.000000  17379.000000  17379.000000
17379.000000
mean       0.028770      3.003683      0.682721      1.425283
0.496987
std        0.167165      2.005771      0.465431      0.639357
0.192556
min        0.000000      0.000000      0.000000      1.000000
0.020000
25%        0.000000      1.000000      0.000000      1.000000
0.340000
50%        0.000000      3.000000      1.000000      1.000000
0.500000
75%        0.000000      5.000000      1.000000      2.000000
0.660000
max        1.000000      6.000000      1.000000      4.000000
1.000000

              atemp           hum     windspeed        casual
registered  \
count  17379.000000  17379.000000  17379.000000  17379.000000
17379.000000
mean       0.475775      0.627229      0.190098     35.676218
```

```
153.786869
std         0.171850      0.192930      0.122340     49.305030
151.357286
min         0.000000      0.000000      0.000000      0.000000
0.000000
25%         0.333300      0.480000      0.104500      4.000000
34.000000
50%         0.484800      0.630000      0.194000     17.000000
115.000000
75%         0.621200      0.780000      0.253700     48.000000
220.000000
max         1.000000      1.000000      0.850700    367.000000
886.000000

                 cnt
count  17379.000000
mean     189.463088
std      181.387599
min        1.000000
25%       40.000000
50%      142.000000
75%      281.000000
max      977.000000
```

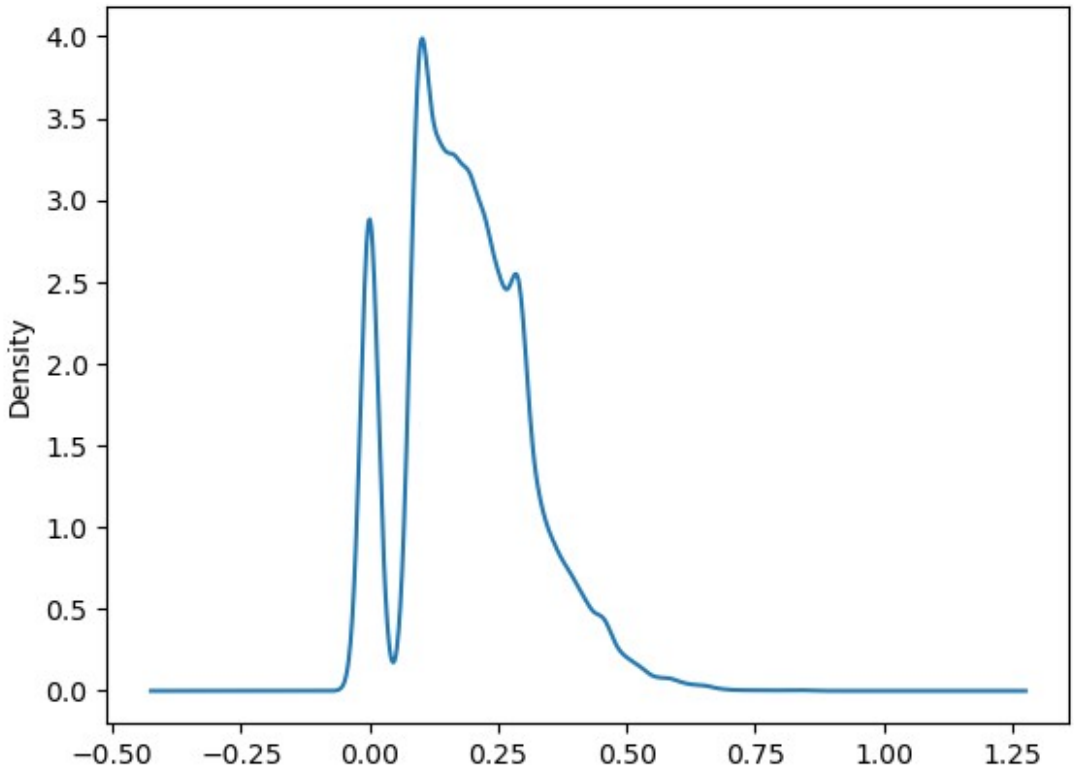
```
# Make density plot for temp. This would give a sense of the
centrality and the spread of the distribution.

df.temp.plot.density(color='red')
plt.show()
```




```
# Boxplot for atemp

# Are there any outliers?

sns.boxplot(df['atemp'])
plt.show()
```

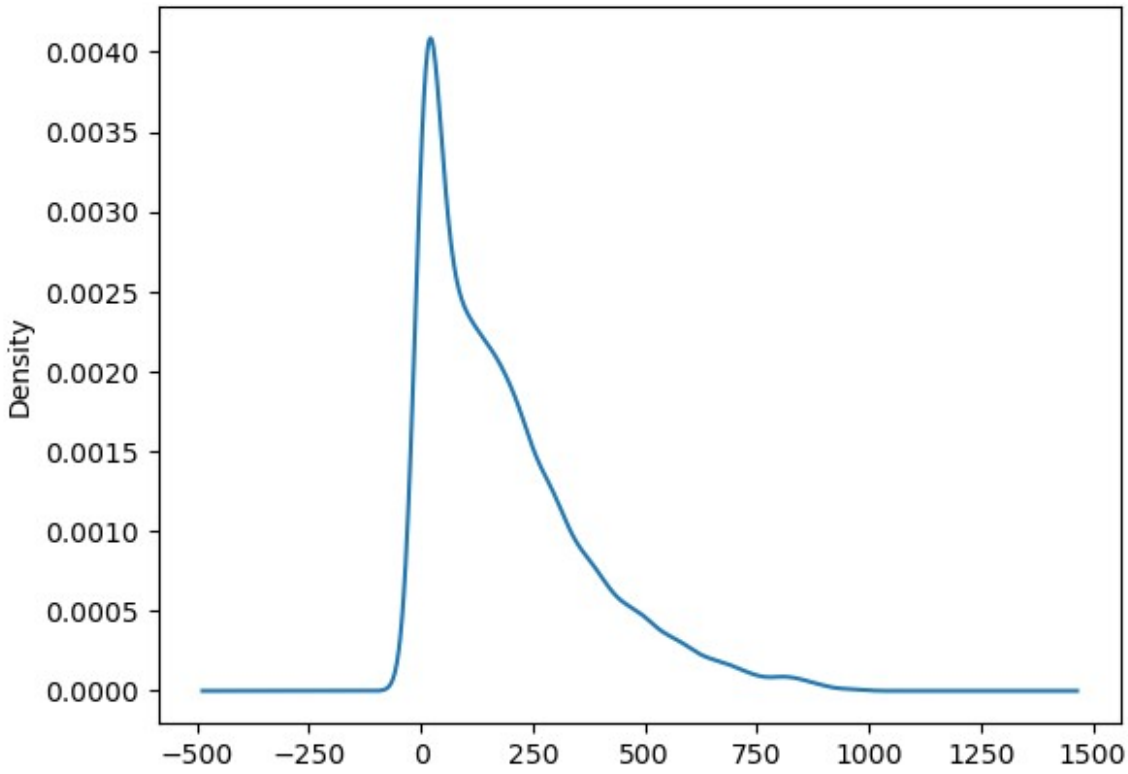



```
# Density plot for windspeed

df.windspeed.plot.density()
plt.show()
```



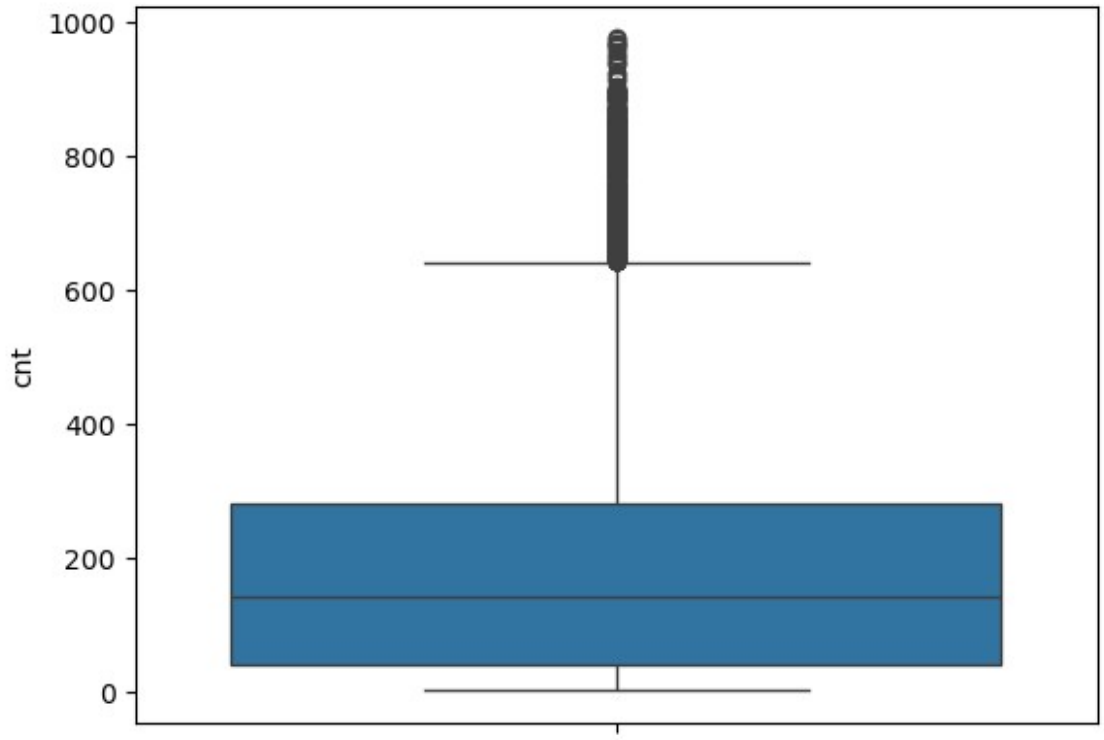

```
# Box and density plot for cnt – this is the variable of interest

# Do you see any outliers in the boxplot?

# Does the density plot provide a similar insight?

df.cnt.plot.density()
```

```
<Axes: ylabel='Density'>
```




```
sns.boxplot(df['cnt'])
```

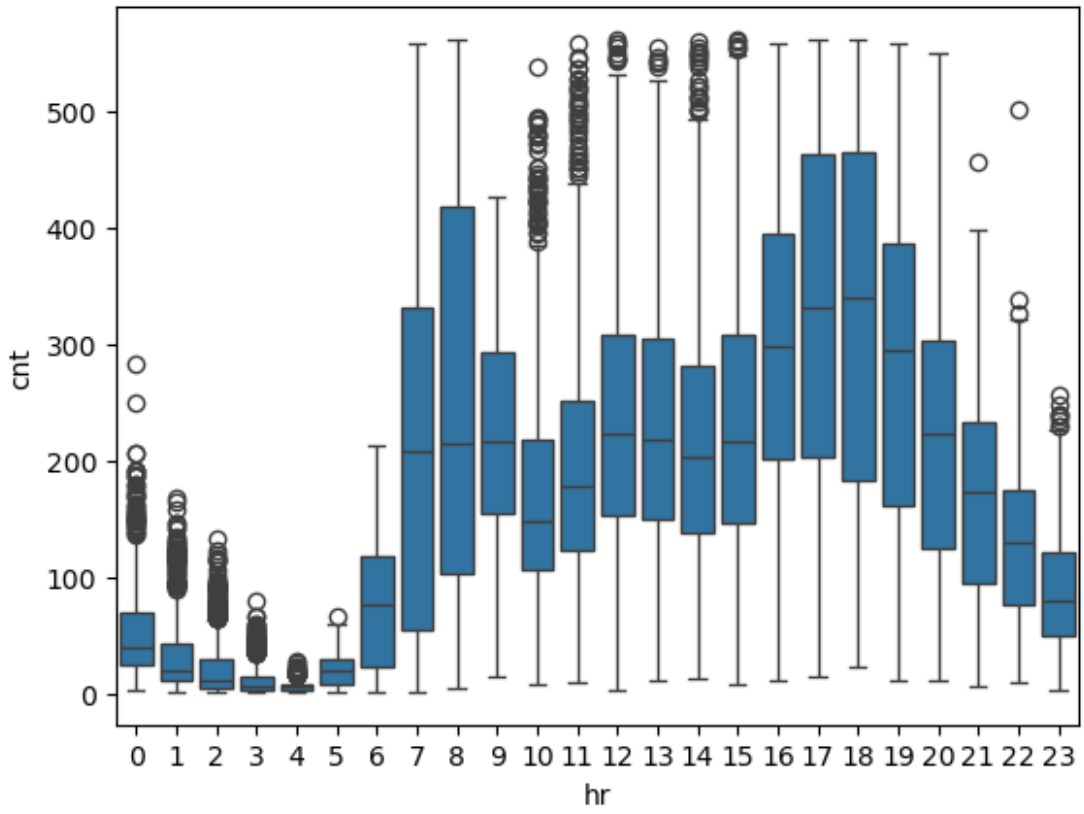
```
<Axes: ylabel='cnt'>
```



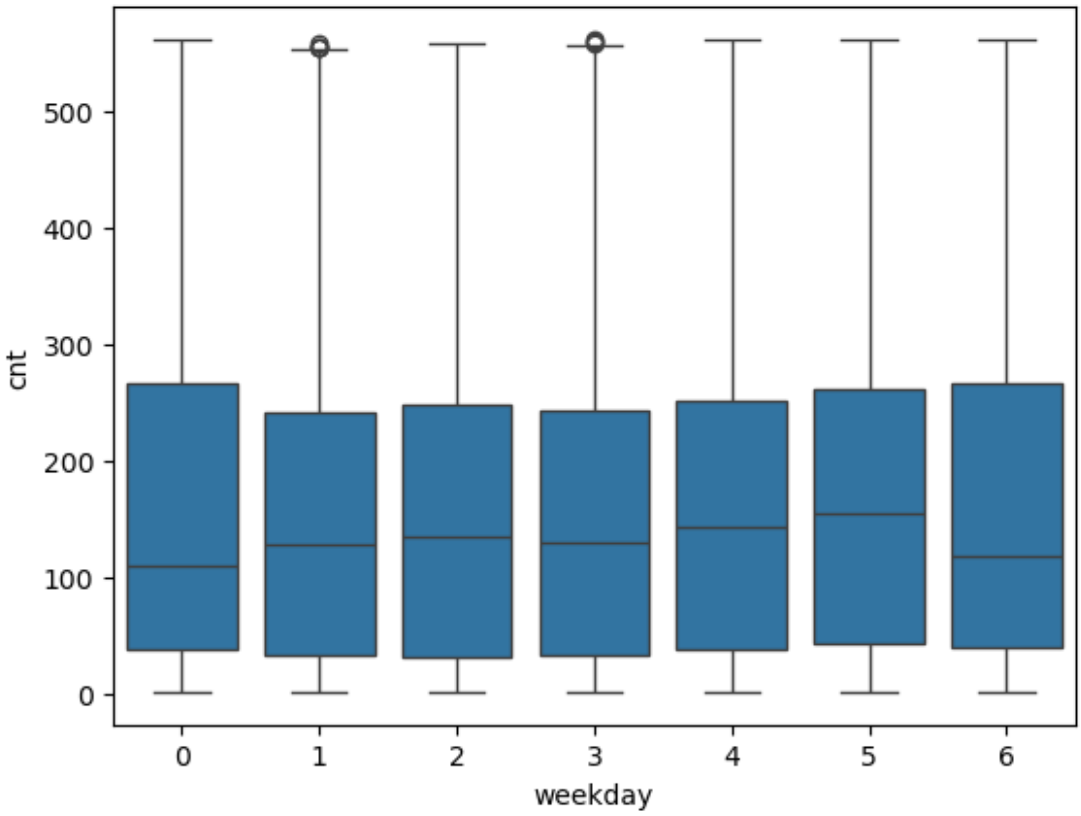

```
inp1.cnt.quantile([0.1, 0.25, 0.5, 0.70, 0.9, 0.95, 0.99])
```

```
0.10      9.00
0.25     40.00
0.50    142.00
0.70    244.00
0.90    451.20
0.95    563.10
0.99    782.22
Name: cnt, dtype: float64
```

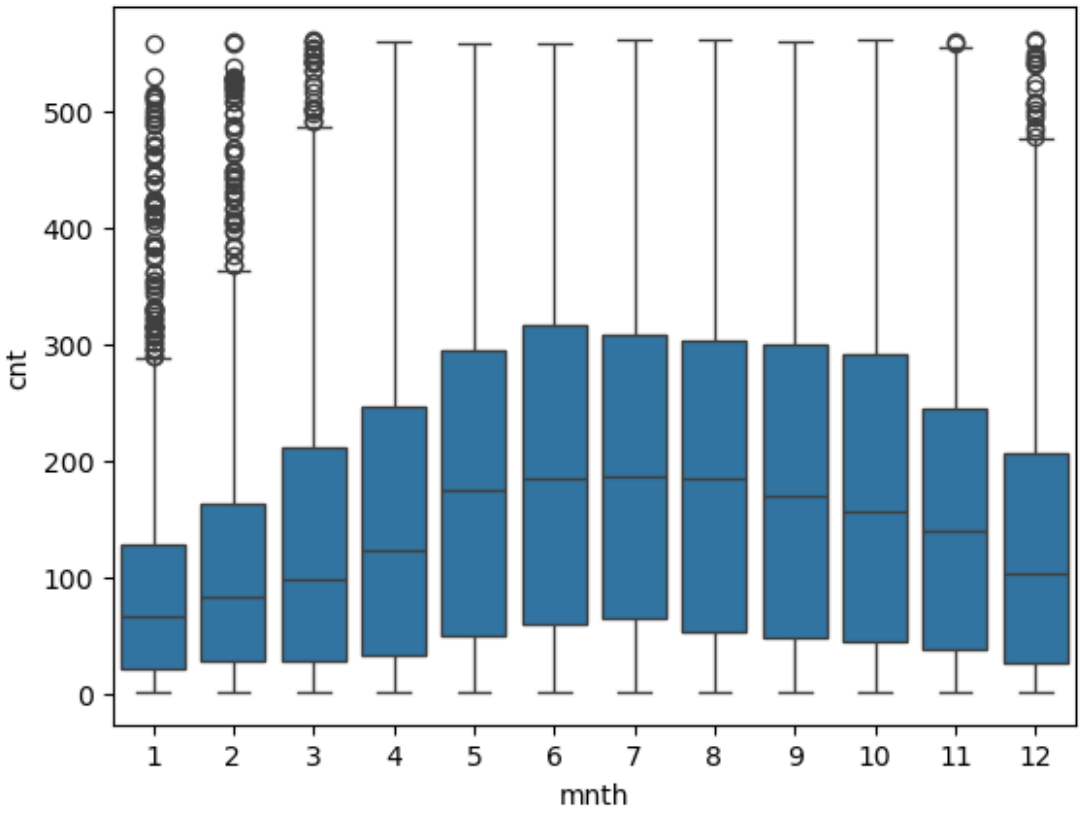
```
# 563 is the 95th percentile – only 5% records have a value higher
than this. Taking this as the cutoff.
inp2 = inp1[inp1.cnt < 563].copy()

# Make boxplot for cnt vs. hour

# What kind of pattern do you see?

sns.boxplot(x='hr',y='cnt',data=inp2)
plt.show()
```



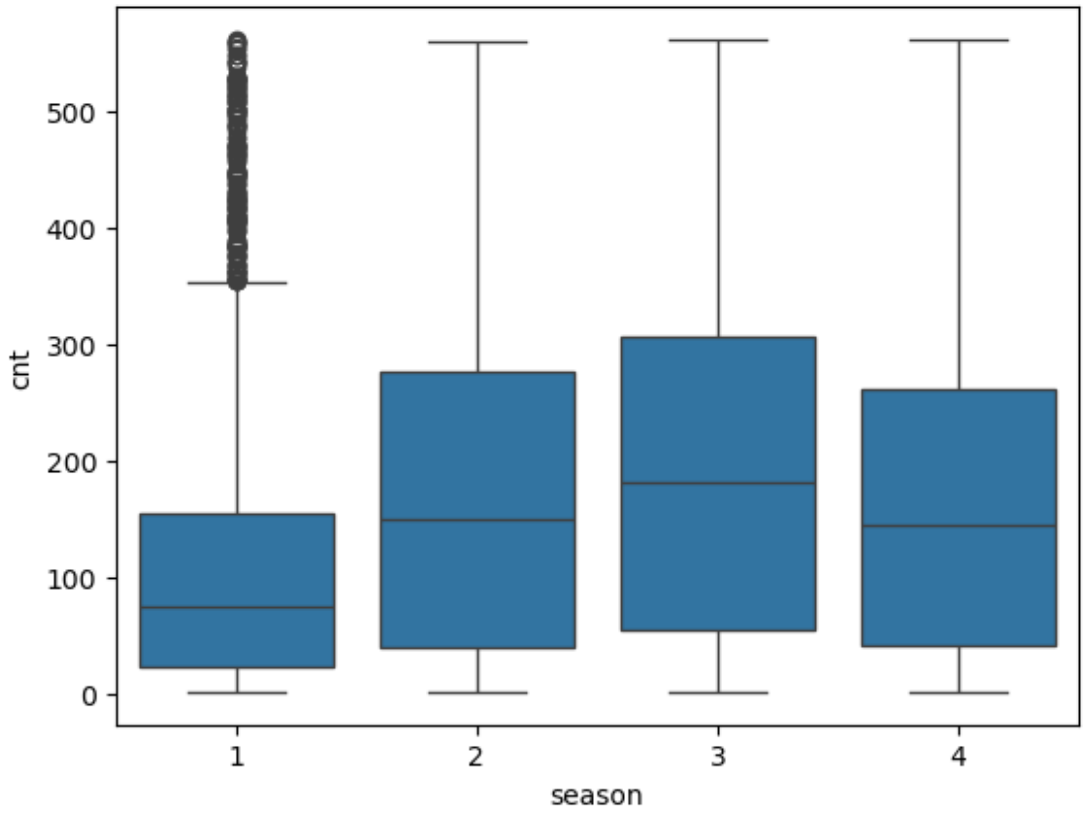

```
# Make boxplot for cnt vs. weekday

# Is there any difference in the rides by days of the week?

sns.boxplot(x='weekday',y='cnt',data= inp2)
plt.show()
```




```
# Make boxplot for cnt vs. month
# Look at the median values. Any month(s) that stand out?

sns.boxplot(x='mnth',y='cnt',data =inp2)
plt.show()
```




```
# 4.  Make boxplot for cnt vs season
# a.  Which season has the highest rides in general? Expected?

sns.boxplot(x='season',y='cnt',data =inp2)
plt.show()
```

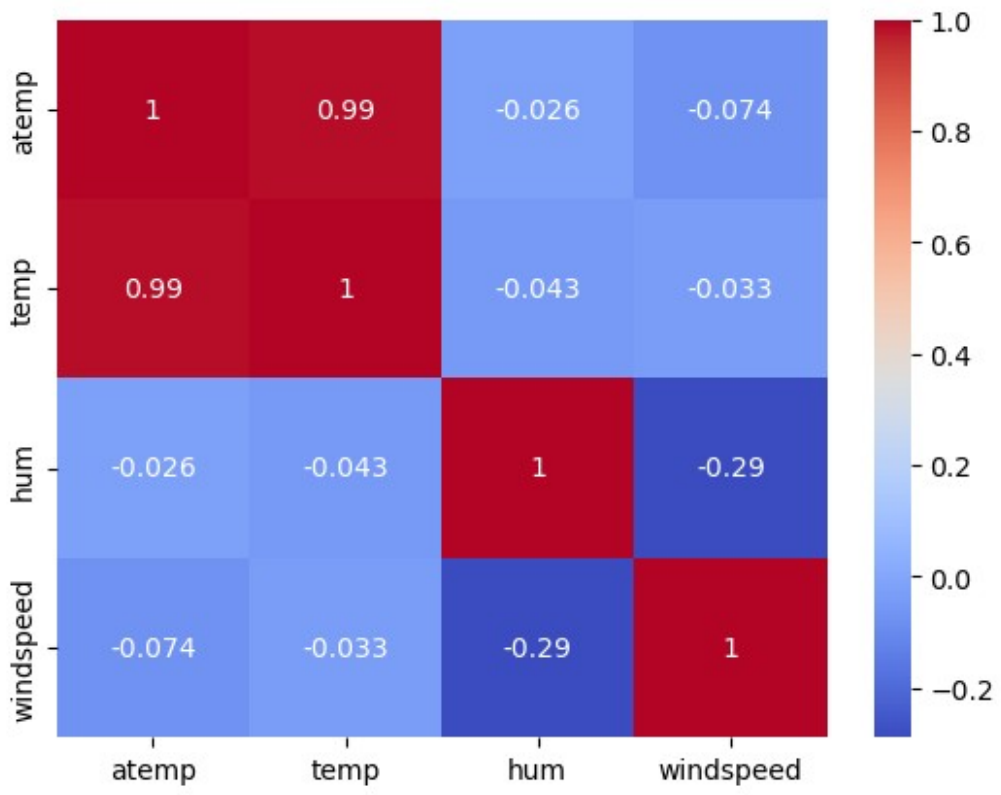



```
# Make a correlation matrix for variables atemp, temp, hum, and
windspeed

# Which variables have the highest correlation?

var = ['atemp', 'temp', 'hum', 'windspeed']

data = inp2[var].corr()
data
```

```
               atemp      temp       hum  windspeed
atemp       1.000000  0.988218 -0.025747  -0.073985
temp        0.988218  1.000000 -0.042603  -0.033209
hum        -0.025747 -0.042603  1.000000  -0.288648
windspeed  -0.073985 -0.033209 -0.288648   1.000000
```

```
sns.heatmap(data,annot=True,cmap='coolwarm')
```

```
<Axes: >
```




```
inp3 = inp2.copy()
inp3.mnth[inp3.mnth.isin([5,6,7,8,9])] = 5
np.unique(inp3.mnth)
```

```
C:\Users\Riya\AppData\Local\Temp\ipykernel_15084\2290649808.py:2:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas
3.0!
You are setting values through chained assignment. Currently this
works in certain cases, but when using Copy-on-Write (which will
become the default behaviour in pandas 3.0) this will never work to
update the original DataFrame or Series, because the intermediate
object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a
DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the
assignment in a single step and ensure this keeps updating the
original `df`.

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
```

```
  inp3.mnth[inp3.mnth.isin([5,6,7,8,9])] = 5
C:\Users\Riya\AppData\Local\Temp\ipykernel_15084\2290649808.py: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
  inp3.mnth[inp3.mnth.isin([5,6,7,8,9])] = 5

array([ 1,  2,  3,  4,  5, 10, 11, 12], dtype=int64)

inp3.hr[inp3.hr.isin([0,1,2,3,4,5])] = 0
inp3.hr[inp3.hr.isin([11,12,13,14,15])] = 11

C:\Users\Riya\AppData\Local\Temp\ipykernel_15084\1263539412.py:1:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas
3.0!
You are setting values through chained assignment. Currently this
works in certain cases, but when using Copy-on-Write (which will
become the default behaviour in pandas 3.0) this will never work to
update the original DataFrame or Series, because the intermediate
object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a
DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the
assignment in a single step and ensure this keeps updating the
original `df`.

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy

  inp3.hr[inp3.hr.isin([0,1,2,3,4,5])] = 0
C:\Users\Riya\AppData\Local\Temp\ipykernel_15084\1263539412.py:1:
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
  inp3.hr[inp3.hr.isin([0,1,2,3,4,5])] = 0
C:\Users\Riya\AppData\Local\Temp\ipykernel_15084\1263539412.py:2:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas
3.0!
You are setting values through chained assignment. Currently this
works in certain cases, but when using Copy-on-Write (which will
become the default behaviour in pandas 3.0) this will never work to
```

```
update the original DataFrame or Series, because the intermediate
object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a
DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the
assignment in a single step and ensure this keeps updating the
original `df`.

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy

  inp3.hr[inp3.hr.isin([11,12,13,14,15])] = 11
C:\Users\Riya\AppData\Local\Temp\ipykernel_15084\1263539412.py: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
  inp3.hr[inp3.hr.isin([11,12,13,14,15])] = 11
```

```
cat_cols = ['season', 'weathersit', 'weekday', 'mnth', 'hr']
inp3 = pd.get_dummies(inp3, columns=cat_cols, drop_first=True)
```

```
---------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
Cell In[206], line 2
      1 cat_cols = ['season', 'weathersit', 'weekday', 'mnth', 'hr']
----> 2 inp3 = pd.get_dummies(inp3, columns=cat_cols, drop_first=True)

File D:\ANACONDA\Lib\site-packages\pandas\core\reshape\
encoding.py:169, in get_dummies(data, prefix, prefix_sep, dummy_na,
columns, sparse, drop_first, dtype)
    167     raise TypeError("Input must be a list-like for parameter
`columns`")
    168 else:
--> 169     data_to_encode = data[columns]
    171 # validate prefixes and separator to avoid silently dropping
cols
    172 def check_len(item, name: str):

File D:\ANACONDA\Lib\site-packages\pandas\core\frame.py:4108, in
DataFrame.__getitem__(self, key)
   4106     if is_iterator(key):
```

```
   4107         key = list(key)
-> 4108     indexer = self.columns._get_indexer_strict(key, "columns")
[1]
   4110 # take() does not accept boolean indexers
   4111 if getattr(indexer, "dtype", None) == bool:

File D:\ANACONDA\Lib\site-packages\pandas\core\indexes\base.py:6200,
in Index._get_indexer_strict(self, key, axis_name)
   6197 else:
   6198     keyarr, indexer, new_indexer =
self._reindex_non_unique(keyarr)
-> 6200 self._raise_if_missing(keyarr, indexer, axis_name)
   6202 keyarr = self.take(indexer)
   6203 if isinstance(key, Index):
   6204     # GH 42790 - Preserve name from an Index

File D:\ANACONDA\Lib\site-packages\pandas\core\indexes\base.py:6249,
in Index._raise_if_missing(self, key, indexer, axis_name)
   6247 if nmissing:
   6248     if nmissing == len(indexer):
-> 6249         raise KeyError(f"None of [{key}] are in the
[{axis_name}]")
   6251     not_found = list(ensure_index(key)[missing_mask.nonzero()
[0]].unique())
   6252     raise KeyError(f"{not_found} not in index")

KeyError: "None of [Index(['season', 'weathersit', 'weekday', 'mnth',
'hr'], dtype='object')] are in the [columns]"


inp3.columns

Index(['yr', 'holiday', 'workingday', 'temp', 'atemp', 'hum',
'windspeed',
       'cnt', 'season_2', 'season_3', 'season_4', 'weathersit_2',
       'weathersit_3', 'weathersit_4', 'weekday_1', 'weekday_2',
'weekday_3',
       'weekday_4', 'weekday_5', 'weekday_6', 'mnth_2', 'mnth_3',
'mnth_4',
       'mnth_5', 'mnth_10', 'mnth_11', 'mnth_12', 'hr_6', 'hr_7',
'hr_8',
       'hr_9', 'hr_10', 'hr_11', 'hr_16', 'hr_17', 'hr_18', 'hr_19',
'hr_20',
       'hr_21', 'hr_22', 'hr_23'],
      dtype='object')

X = inp3.drop(['cnt'],axis=1)
y = inp3['cnt']

y.shape
```

```
(16502,)
```

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,train_size =
0.7,random_state = 2592)
X_train
```

```
       yr  holiday  workingday  temp   atemp   hum  windspeed
season_2  \
7479    0        0           0  0.34  0.3333  0.66     0.1343
False
17234   1        1           0  0.28  0.2727  0.65     0.2537
False
16489   1        0           0  0.26  0.2424  0.41     0.2537
False
14060   1        0           1  0.68  0.6364  0.79     0.2537
False
1373    0        0           1  0.40  0.4091  0.43     0.1940
False
...    ..      ...         ...   ...     ...   ...        ...       ..
.
10188   1        0           1  0.28  0.2576  0.57     0.3582
False
10398   1        0           1  0.52  0.5000  0.68     0.0000
False
10374   1        0           1  0.48  0.4697  0.82     0.2836
False
1332    0        0           1  0.44  0.4394  0.88     0.6119
False
7957    0        0           0  0.24  0.2576  0.70     0.0896
False

       season_3  season_4  ...  hr_10  hr_11  hr_16  hr_17  hr_18
hr_19  \
7479      False      True  ...  False  False  False  False  False
False
17234     False     False  ...  False  False  False  False  False
False
16489     False      True  ...  False  False  False  False  False
True
14060      True     False  ...  False  False  False  False  False
False
1373      False     False  ...  False   True  False  False  False
False
...         ...       ...  ...    ...    ...    ...    ...    ...    .
..
10188     False     False  ...  False   True  False  False  False
False
10398     False     False  ...  False  False  False  False  False
```

```
False
10374     False     False  ...  False  False  False  False  False
False
1332      False     False  ...  False  False  False  False  False
True
7957      False      True  ...  False  False  False  False  False
False

       hr_20  hr_21  hr_22  hr_23
7479   False  False  False  False
17234  False  False  False   True
16489  False  False  False  False
14060  False  False  False  False
1373   False  False  False  False
...      ...    ...    ...    ...
10188  False  False  False  False
10398  False  False  False  False
10374  False  False  False  False
1332   False  False  False  False
7957   False  False  False  False

[11551 rows x 40 columns]

X_train.shape,X_test.shape,y_train.shape,y_test.shape

((11551, 40), (4951, 40), (11551,), (4951,))

from sklearn.linear_model import LinearRegression
model = LinearRegression()

model.fit(X_train,y_train)

LinearRegression()

model.intercept_

-85.34565189161111

model.coef_

array([ 5.88131405e+01,  1.54846235e+15,  1.54846235e+15,
1.21424167e+02,
        8.95059633e+01, -5.49123777e+01, -2.89339860e+01,
3.02467387e+01,
        1.49240224e+01,  6.47797944e+01, -6.10595929e+00, -
5.78196448e+01,
       -3.62211305e+01, -1.54846235e+15, -1.54846235e+15, -
1.54846235e+15,
       -1.54846235e+15, -1.54846235e+15,  1.16967137e+01,
4.44270017e+00,
        8.26827194e+00, -1.78838945e+00,  1.20228229e+01, -
```

```
6.24811413e+00,
       -1.23800876e+01, -1.01473593e+01,  6.04486775e+01,
1.91553561e+02,
        2.49069985e+02,  1.90762248e+02,  1.36902470e+02,
1.76664190e+02,
        2.45215392e+02,  3.05135282e+02,  2.93362565e+02,
2.49725929e+02,
        1.84166281e+02,  1.34893453e+02,  9.61822333e+01,
5.77821906e+01])

pred = model.predict(X_test)
pred

array([302.84136074, 211.16848216, 210.78014841, ...,  89.62584853,
       124.05681852,  95.45936162])

from sklearn.metrics import
mean_absolute_percentage_error,mean_absolute_error,mean_squared_error

mean_absolute_percentage_error(y_test,pred)

2.4101818446957464

mean_absolute_error(y_test,pred)

62.6395381101166

mean_squared_error(y_test,pred)

6748.009232297059
```