In [443]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
#%matplotlib notebook
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

1. Load the data file.

In [356]:

```
df=pd.read_csv("hour.csv")
```

In [357]:

```
df.head()
```

Out[357]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0

In [358]:

```
df.shape
```

Out[358]:

```
(17379, 17)
```

In [359]:

```
df.describe()
```

Out[359]:

	instant	season	yr	mnth	hr	holiday	weekday	workingday	we
count	17379.0000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379
mean	8690.0000	2.501640	0.502561	6.537775	11.546752	0.028770	3.003683	0.682721	1
std	5017.0295	1.106918	0.500008	3.438776	6.914405	0.167165	2.005771	0.465431	0
min	1.0000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1
25%	4345.5000	2.000000	0.000000	4.000000	6.000000	0.000000	1.000000	0.000000	1
50%	8690.0000	3.000000	1.000000	7.000000	12.000000	0.000000	3.000000	1.000000	1
75%	13034.5000	3.000000	1.000000	10.000000	18.000000	0.000000	5.000000	1.000000	2
max	17379.0000	4.000000	1.000000	12.000000	23.000000	1.000000	6.000000	1.000000	4

In [360]:

```
df.dtypes
```

Out[360]:

```
instant         int64
dteday         object
season          int64
yr              int64
mnth            int64
hr              int64
holiday         int64
weekday         int64
workingday      int64
weathersit      int64
temp          float64
atemp         float64
hum           float64
windspeed     float64
casual          int64
registered      int64
cnt             int64
dtype: object
```

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

In [361]:

```
df_missing=df.isna()
```

In [362]:

```
for c in df_missing.columns.values.tolist():
    print (c)
    print(df_missing[c].value_counts())
```

```
instant
False    17379
Name: instant, dtype: int64
dteday
False    17379
Name: dteday, dtype: int64
season
False    17379
Name: season, dtype: int64
yr
False    17379
Name: yr, dtype: int64
mnth
False    17379
Name: mnth, dtype: int64
hr
False    17379
Name: hr, dtype: int64
holiday
False    17379
Name: holiday, dtype: int64
weekday
False    17379
Name: weekday, dtype: int64
workingday
False    17379
Name: workingday, dtype: int64
weathersit
False    17379
Name: weathersit, dtype: int64
temp
False    17379
```

```
Name: temp, dtype: int64
atemp
False    17379
Name: atemp, dtype: int64
hum
False    17379
Name: hum, dtype: int64
windspeed
False    17379
Name: windspeed, dtype: int64
casual
False    17379
Name: casual, dtype: int64
registered
False    17379
Name: registered, dtype: int64
cnt
False    17379
Name: cnt, dtype: int64
```

There doesn't seem to be any NA values to drop

In [363]:

```
df.shape
```

Out[363]:

```
(17379, 17)
```

In [364]:

```
df.dropna(inplace=True)
df.shape # Shape is the same
```

Out[364]:

```
(17379, 17)
```

3. Sanity checks:

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

In [365]:

```
condition=df["registered"]+df["casual"]!=df["cnt"]

df[condition].shape # no junk rows to drop
```

Out[365]:

```
(0, 17)
```

3.2 Month values should be 1-12 only

In [366]:

```
df["mnth"].describe() # min is 1, max is 12
```

Out[366]:

```
count    17379.000000
mean         6.537775
std          3.438776
min          1.000000
25%          4.000000
50%          7.000000
75%         10.000000
max         12.000000
Name: mnth, dtype: float64
```

```
Name: mnth, dtype: float64
```

In [367]:

```
# double checking

condition1=(df["mnth"]<1) | (df["mnth"]>12)

df[condition1] # no records beyond this range
```

Out[367]:

instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	r

3.3 Hour values should be 0-23

In [368]:

```
condn=(df["hr"]<0) |(df["hr"]>23)
df[condn] # no records outside 0-23
```

Out[368]:

instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	r

Step 4

The variables ‘casual’ and ‘registered’ are redundant and need to be dropped.

In [369]:

```
inp1=df.drop(["casual","registered"], inplace=False,axis=1)
```

In [370]:

```
inp1.shape # columns were dropped
```

Out[370]:

```
(17379, 15)
```

‘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.

In [371]:

```
inp1.drop(["instant","dteday"],inplace=True,axis=1)
```

In [372]:

```
inp1.head()
```

Out[372]:

	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt
0	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	16
1	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	40
2	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	32
3	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	13
4	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	1

5. Univariate Analysis

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

In [373]:

```
inp1.describe()
```

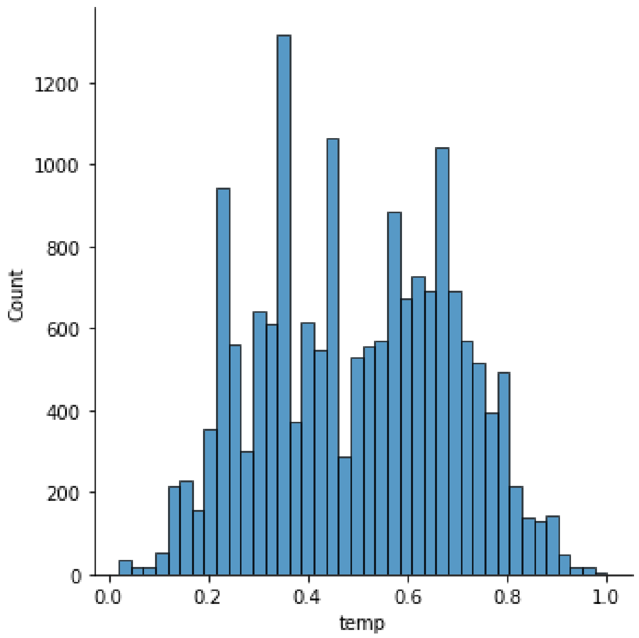
Out[373]:

	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	173
mean	2.501640	0.502561	6.537775	11.546752	0.028770	3.003683	0.682721	1.425283	
std	1.106918	0.500008	3.438776	6.914405	0.167165	2.005771	0.465431	0.639357	
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	
25%	2.000000	0.000000	4.000000	6.000000	0.000000	1.000000	0.000000	1.000000	
50%	3.000000	1.000000	7.000000	12.000000	0.000000	3.000000	1.000000	1.000000	
75%	3.000000	1.000000	10.000000	18.000000	0.000000	5.000000	1.000000	2.000000	
max	4.000000	1.000000	12.000000	23.000000	1.000000	6.000000	1.000000	4.000000	

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

In [374]:

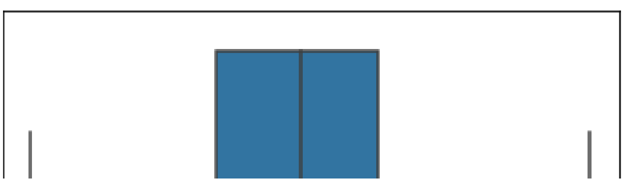
```
sns.displot(inp1["temp"])
plt.show()
```

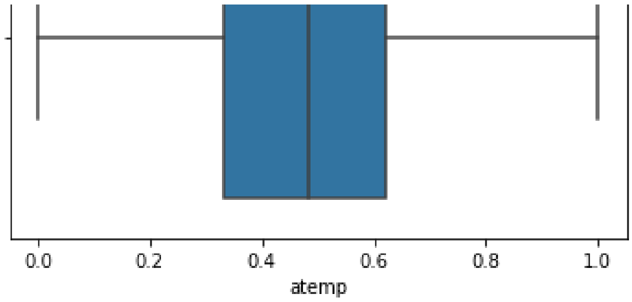


Boxplot for atemp

In [375]:

```
sns.boxplot(x="atemp",data=inp1)
plt.show()
```



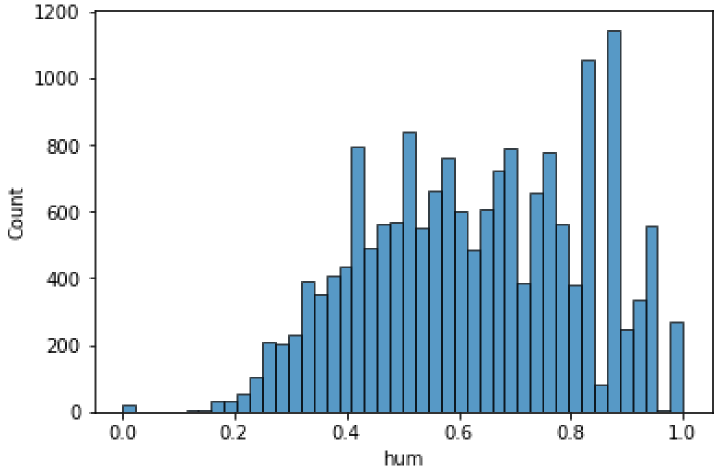


Are there any outliers?
No, there don't seem to be any extreme outliers

Histogram for hum

In [376]:

```
sns.histplot(inp1["hum"])
plt.show()
```

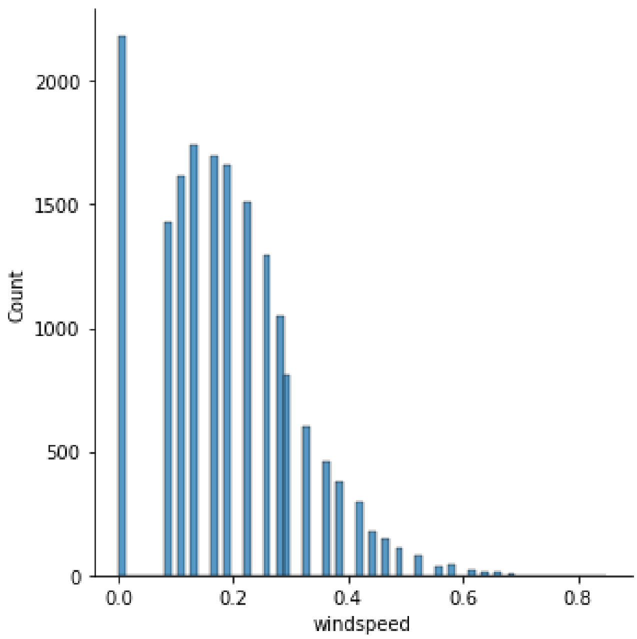


Do you detect any abnormally high values?
There are around 200 rows that report an humidity of 1.0 (100 %), which seems strange but could be valid

Density plot for windspeed

In [377]:

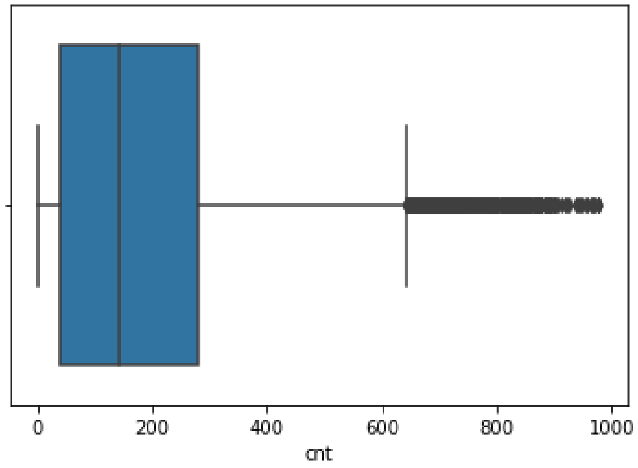
```
sns.displot(x="windspeed",data=inp1)
plt.show()
```



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

In [378]:

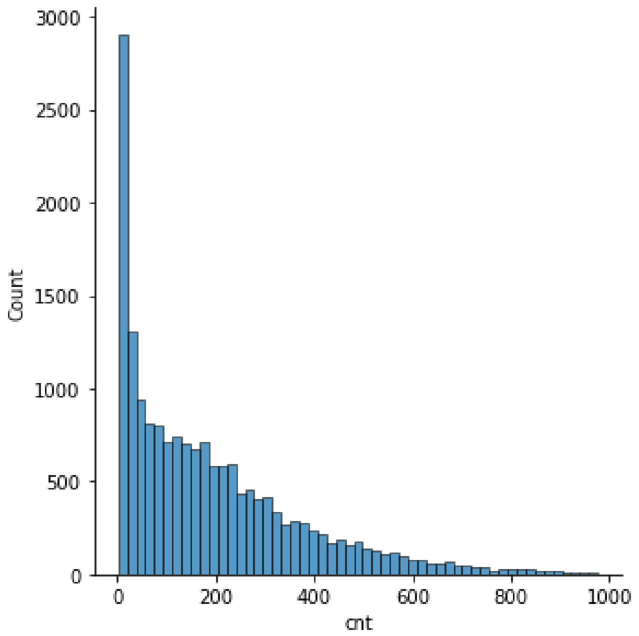
```
sns.boxplot(x="cnt",data=inp1)
plt.show()
```



Do you see any outliers in the boxplot?
Yes, there are many outliers ranging from the 600 to 1000 + range

In [379]:

```
sns.displot(x="cnt",data=inp1)
plt.show()
```



Does the density plot provide a similar insight?
Yes, it also indicates outliers at the same range specified above

6. Outlier treatment

Cnt looks like some hours have rather high values. You'll need to treat these outliers so that they don't skew the analysis and the model.

Find out the following percentiles: 10, 25, 50, 75, 90, 95, 99

In [380]:

```
inp1["cnt"].quantile([0.10,0.25,0.50,0.75,0.90,0.95,0.99])
```

Out[380]:

```
0.10      9.00
0.25     40.00
0.50    142.00
0.75    281.00
0.90    451.20
0.95    563.10
0.99    782.22
Name: cnt, dtype: float64
```

Decide the cutoff percentile and drop records with values higher than the cutoff. Name the new dataframe as inp2.

In [381]:

```
# cut off percentile is 90
condn1=inp1["cnt"]>inp1["cnt"].quantile([0.90]).iloc[0]
inp2=inp1.drop(df[condn1].index,inplace=False,axis=0)
```

In [382]:

```
inp1.shape, inp2.shape ## rows with cnt beyond the 90th percentile have been dropped
```

Out[382]:

```
((17379, 13), (15641, 13))
```

7. Bivariate analysis

Make boxplot for cnt vs. hour

In [383]:

```
inp2["hr"].dtype # need to convert hour to strings
```

Out[383]:

```
dtype('int64')
```

In [384]:

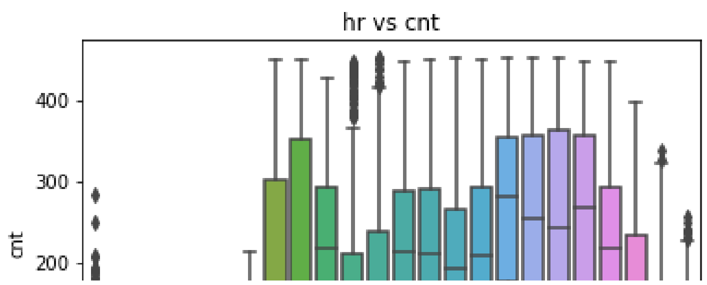
```
inp3=inp2[["hr","cnt"]]
inp3["hr"]=inp3["hr"].astype("str")
```

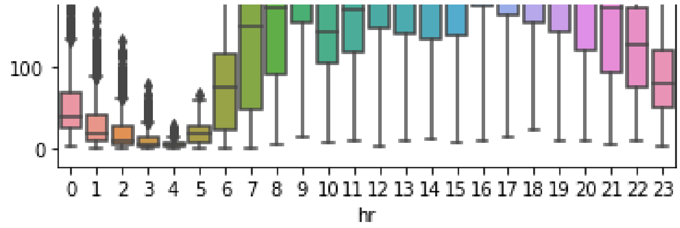
```
C:\Users\skv08\AppData\Local\Temp/ipykernel_3520/513430221.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  inp3["hr"]=inp3["hr"].astype("str")
```

In [385]:

```
sns.boxplot(x="hr",y="cnt",data=inp3)
plt.title("hr vs cnt")
plt.show()
```





What kind of pattern do you see?
The count peaks at around 6 to 9 in the morning, and 4 to 8 in the evening.
This makes sense since people would probably exercise early in the morning or commute after work

Make boxplot for cnt vs. weekday

In [386]:

```
inp3=inp2[["weekday","cnt"]]
```

In [387]:

```
inp3["weekday"].unique()
```

Out[387]:

```
array([6, 0, 1, 2, 3, 4, 5], dtype=int64)
```

In [388]:

```
day_code_dict={6:"Sunday",0:"Monday",1:"Tues",2:"Wed",3:"Thurs",4:"Fri",5:"Sat"}
```

In [389]:

```
inp3["weekday"]=inp3["weekday"].map(day_code_dict)
```

```
C:\Users\skv08\AppData\Local\Temp/ipykernel_3520/939504834.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  inp3["weekday"]=inp3["weekday"].map(day_code_dict)
```

In [390]:

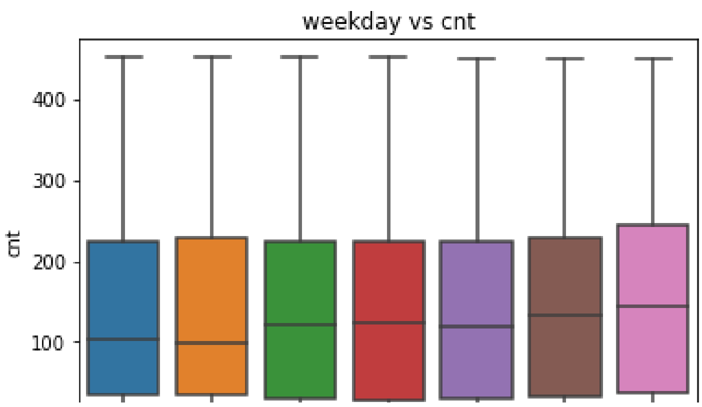
```
inp3["weekday"].unique()
```

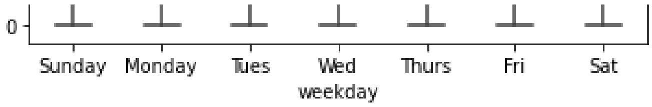
Out[390]:

```
array(['Sunday', 'Monday', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
      dtype=object)
```

In [391]:

```
sns.boxplot(x="weekday",y="cnt",data=inp3)
plt.title("weekday vs cnt")
plt.show()
```





Is there any difference in the rides by days of the week?
Saturday seems to have the highest median of rides, but the ride count seems consistent throughout the week

Make boxplot for cnt vs. month

In [392]:

```
inp2[["cnt","mnth"]].dtypes # both are ints
```

Out[392]:

```
cnt      int64
mnth     int64
dtype: object
```

In [393]:

```
inp3=inp2[["cnt","mnth"]]
```

In [394]:

```
inp3["mnth"].unique().tolist()  # Jan=1,...Dec=12
```

Out[394]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

In [395]:

```
mnth_list=["Jan","Feb","March","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
mnth_mapping_dict=dict(zip(inp3["mnth"].unique().tolist(),mnth_list))
```

In [396]:

```
mnth_mapping_dict
```

Out[396]:

```
{1: 'Jan',
 2: 'Feb',
 3: 'March',
 4: 'Apr',
 5: 'May',
 6: 'Jun',
 7: 'Jul',
 8: 'Aug',
 9: 'Sep',
 10: 'Oct',
 11: 'Nov',
 12: 'Dec'}
```

In [397]:

```
inp3["mnth"]=inp3["mnth"].map(mnth_mapping_dict)
```

```
C:\Users\skv08\AppData\Local\Temp/ipykernel_3520/1679586090.py:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  inp3["mnth"]=inp3["mnth"].map(mnth_mapping_dict)
```

In [398]:

```
inp3["mnth"].unique() # values have been changed
```

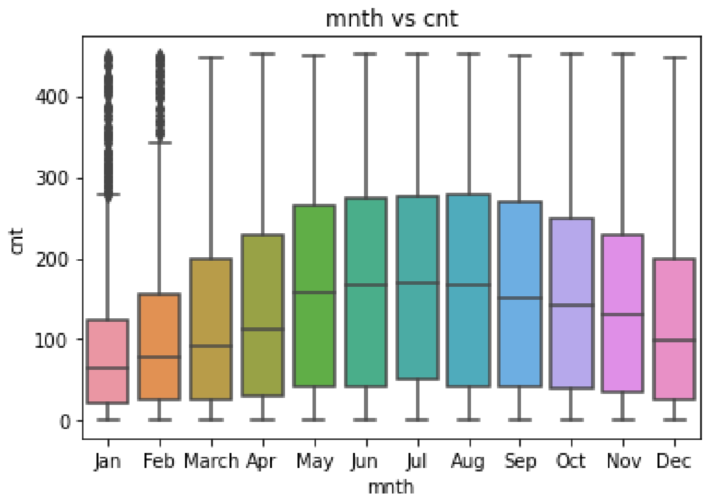
```
inp3["mnth"].unique() # values have been changed
```

Out[398]:

```
array(['Jan', 'Feb', 'March', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
       'Oct', 'Nov', 'Dec'], dtype=object)
```

In [399]:

```
sns.boxplot(x="mnth",y="cnt",data=inp3)
plt.title("mnth vs cnt")
plt.show()
```



Look at the median values. Any month(s) that stand out?
Yes, The middle of the year (June, July, Aug) seem to have the highest median counts

Make boxplot for cnt vs. season

In [400]:

```
inp2["season"].unique() #1:spring, 2:summer, 3:fall, 4:winter
```

Out[400]:

```
array([1, 2, 3, 4], dtype=int64)
```

In [401]:

```
seasons_dict={1:"spring", 2:"summer", 3:"fall", 4:"winter"}
```

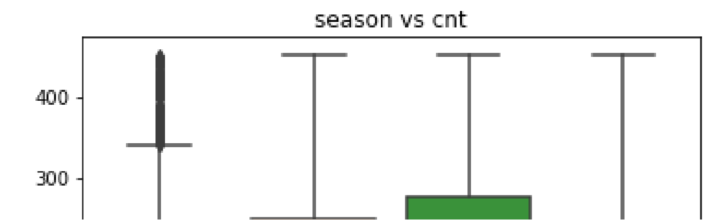
In [402]:

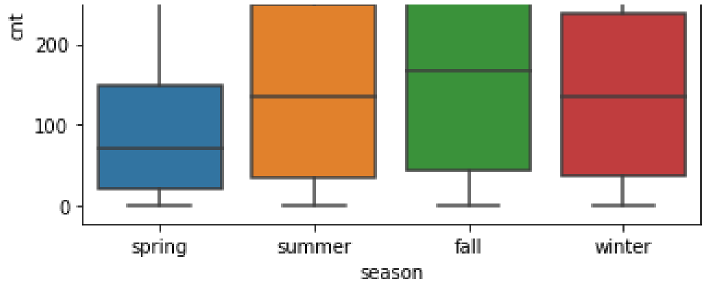
```
inp3=inp2[["cnt","season"]]
inp3["season"]=inp3["season"].map(seasons_dict)

sns.boxplot(x="season",y='cnt',data=inp3)
plt.title("season vs cnt")
plt.show()
```

```
C:\Users\skv08\AppData\Local\Temp/ipykernel_3520/157664810.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  inp3["season"]=inp3["season"].map(seasons_dict)
```





Which season has the highest rides in general? Expected?
The fall season has the highest rides in general. This makes sense as the weather would be good for biking.
I was surprised that the summer and spring seasons weren't the highest. In fact, the spring season is even less than the
winter season

Make a bar plot with the median value of cnt for each hr

In [403]:

```
inp3=inp2[["cnt","hr"]]
inp3["hr"]=inp3["hr"].astype("int")
```

```
C:\Users\skv08\AppData\Local\Temp/ipykernel_3520/712403034.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  inp3["hr"]=inp3["hr"].astype("int")
```

In [404]:

```
grouped_inp3=inp3.groupby("hr").median()
```

In [405]:

```
inp3["hr"].dtype
```

Out[405]:

```
dtype('int32')
```

In [406]:

```
#grouped_inp3.index=grouped_inp3.index.map(str)
```

In [407]:

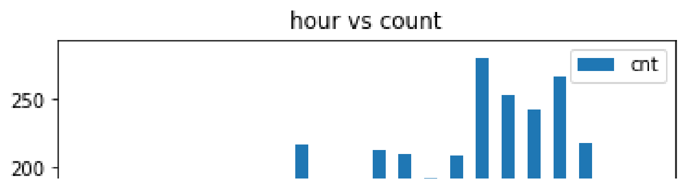
```
grouped_inp3.index
```

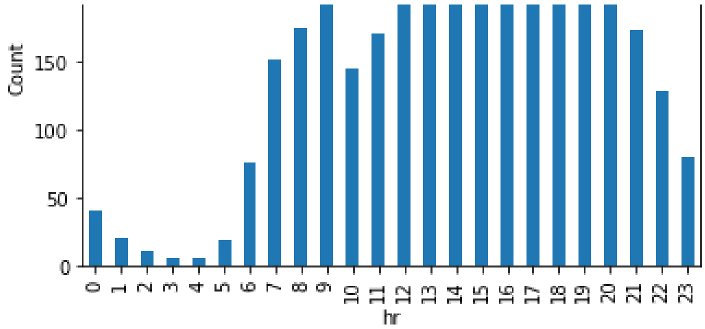
Out[407]:

```
Int64Index([ 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', name='hr')
```

In [408]:

```
grouped_inp3.plot(kind="bar")
plt.title("hour vs count")
plt.ylabel("Count")
plt.show()
```





Does this paint a different picture from the box plot?
No, like the box plot indicated, the peak hours seem to be in the evening (4-7 pm) and also morning right before work hours 7 to 9. This makes sense as people would be commuting to and from work

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

In [409]:

```
inp3=inp2[["atemp","temp", "hum","windspeed"]]
```

In [410]:

```
inp3.head()
```

Out[410]:

	atemp	temp	hum	windspeed
0	0.2879	0.24	0.81	0.0
1	0.2727	0.22	0.80	0.0
2	0.2727	0.22	0.80	0.0
3	0.2879	0.24	0.75	0.0
4	0.2879	0.24	0.75	0.0

In [411]:

```
inp3.corr()
```

Out[411]:

	atemp	temp	hum	windspeed
atemp	1.000000	0.988420	0.001513	-0.086428
temp	0.988420	1.000000	-0.013957	-0.044807
hum	0.001513	-0.013957	1.000000	-0.285215
windspeed	-0.086428	-0.044807	-0.285215	1.000000

Which variables have the highest correlation?
Temp and atemp have the highest correlation (0.98). This makes sense by definition

8. Data preprocessing

8.1 Treating mnth column

8.1.1 For values 5,6,7,8,9,10, replace with a single value 5. This is because these have very similar values for cnt.

In [412]:

```
inp2["mnth"].replace(list(range(5,11)),5,inplace=True)
```

In [413]:

```
inp2["mnth"].unique() # 5 through 10 have been subsituted with 5
```

Out[413]:

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

8.1.2 Get dummies for the updated 6 mnth values

In [414]:

```
inp2_mnth_dummies=pd.get_dummies(inp2["mnth"])
```

In [415]:

```
inp2_concat=pd.concat([inp2,inp2_mnth_dummies],axis=1)
```

In [416]:

```
inp2_concat.shape # columns have been added
inp2_concat.drop(columns=["mnth"],inplace=True) # dropping original mnth col
```

In [417]:

```
inp2_concat.rename(columns={1: "Jan",2:"Feb",3:"Mar",4:"Apr",5:"M-Oct",11:"Nov",12:"Dec"
},inplace=True)
```

In [418]:

```
inp2_concat.head()
```

Out[418]:

	season	yr	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt	Jan	Feb	Mar	Apr	M-Oct
0	1	0	0	0	6	0	1	0.24	0.2879	0.81	0.0	16	1	0	0	0	0
1	1	0	1	0	6	0	1	0.22	0.2727	0.80	0.0	40	1	0	0	0	0
2	1	0	2	0	6	0	1	0.22	0.2727	0.80	0.0	32	1	0	0	0	0
3	1	0	3	0	6	0	1	0.24	0.2879	0.75	0.0	13	1	0	0	0	0
4	1	0	4	0	6	0	1	0.24	0.2879	0.75	0.0	1	1	0	0	0	0

8.2 Treating hr column

8.2.1 Create new mapping: 0-5: 0, 11-15: 11; other values are untouched.

In [419]:

```
inp2["hr"].unique()
```

Out[419]:

```
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)
```

In [420]:

```
inp2["hr"].replace(list(range(0,6)),0,inplace=True)
inp2["hr"].replace(list(range(11,16)),11,inplace=True)
```

In [421]:

```
hr_dummies=pd.get_dummies(inp2["hr"])
```

In [422]:

```
inp2_concat=pd.concat([inp2_concat,hr_dummies],axis=1)
```

In [423]:

```
inp2_concat.drop(columns="hr",inplace=True)
```

In [424]:

```
# renaming columns
col_list=inp2_concat.columns.tolist()

col_list=col_list[-15:]# The columns with the hour values, 0, 1, 2...23
```

In [425]:

```
renamed_hrs_dict={}
for col_name in col_list:
    renamed_hrs_dict[col_name]="Hour:" + " "+ str(col_name)

# renaming clustered hour ranges 0-5, 11-15
renamed_hrs_dict[0]="Hours: 0-5"
renamed_hrs_dict[11]="Hours: 11-15"
```

In [426]:

```
renamed_hrs_dict
```

Out[426]:

```
{0: 'Hours: 0-5',
 6: 'Hour: 6',
 7: 'Hour: 7',
 8: 'Hour: 8',
 9: 'Hour: 9',
 10: 'Hour: 10',
 11: 'Hours: 11-15',
 16: 'Hour: 16',
 17: 'Hour: 17',
 18: 'Hour: 18',
 19: 'Hour: 19',
 20: 'Hour: 20',
 21: 'Hour: 21',
 22: 'Hour: 22',
 23: 'Hour: 23'}
```

In [427]:

```
inp2_concat.rename(renamed_hrs_dict,inplace=True,axis=1)
```

In [428]:

```
inp2_concat.columns
```

Out[428]:

```
Index(['season', 'yr', 'holiday', 'weekday', 'workingday', 'weathersit',
       'temp', 'atemp', 'hum', 'windspeed', 'cnt', 'Jan', 'Feb', 'Mar', 'Apr',
       'M-Oct', 'Nov', 'Dec', 'Hours: 0-5', 'Hour: 6', 'Hour: 7', 'Hour: 8',
       'Hour: 9', 'Hour: 10', 'Hours: 11-15', 'Hour: 16', 'Hour: 17',
       'Hour: 18', 'Hour: 19', 'Hour: 20', 'Hour: 21', 'Hour: 22', 'Hour: 23'],
      dtype='object')
```

8.3 Get dummy columns for season, weathersit, weekday as well.

In [429]:

```
inp2=inp2_concat
inp2[["season","weathersit","weekday"]].dtypes # need to convert to object first before g
etting dummy variables
```

Out[429]:

```
season        int64
weathersit    int64
weekday       int64
dtype: object
```

In [430]:

```
inp2[["season","weathersit","weekday"]]=inp2[["season","weathersit","weekday"]].astype("
str")
```

In [431]:

```
df_dummies=pd.get_dummies(inp2[["season","weathersit","weekday"]])
```

In [432]:

```
# concatenating dummy columns back to inp2
inp2=pd.concat([inp2,df_dummies],axis=1)
inp2.drop(columns=["season","weathersit","weekday"],inplace=True)
```

9. Train test split: Apply 70-30 split.

In [435]:

```
df_train, df_test=train_test_split(inp2,test_size=0.3,random_state=0)
```

In [436]:

```
df_train.shape,df_test.shape
```

Out[436]:

```
((10948, 45), (4693, 45))
```

10. Separate X and Y for df_train and df_test

In [451]:

```
X_train=df_train.drop(columns="cnt",inplace=False)
y_train=df_train["cnt"]
X_test=df_test.drop(columns="cnt",inplace=False)
y_test=df_test["cnt"]
```

10 . Model building

Use linear regression as the technique

In [444]:

```
lm=LinearRegression()
```

In [445]:

```
lm.fit(X_train,y_train)
```

Out[445]:

```
LinearRegression()
```

Report the R2 on the train set

In [447]:

```
lm.score(X_train,y_train)
```

Out[447]:

```
0.6732599447846925
```

11. Make predictions on test set and report R2.

In [448]:

```
yhat=lm.predict(X_test)
yhat
```

Out[448]:

```
array([130.125, 165.25 ,  48.375, ..., 224.25 , 188.875, 184.125])
```

In [453]:

```
lm.score(X_test,y_test)
```

Out[453]:

```
0.6595175436957077
```