===

Bike Sharing Dataset

Hadi Fanaee-T

https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset# (https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset#)

Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto INESC Porto, Campus da FEUP Rua Dr. Roberto Frias, 378 4200 - 465 Porto, Portugal

==

Background

Bike sharing systems are new generation of traditional bike rentals where whole process from membership, rental and return back has become automatic. Through these systems, user is able to easily rent a bike from a particular position and return back at another position. Currently, there are about over 500 bike-sharing programs around the world which is composed of over 500 thousands bicycles. Today, there exists great interest in these systems due to their important role in traffic, environmental and health issues.

Apart from interesting real world applications of bike sharing systems, the characteristics of data being generated by these systems make them attractive for the research. Opposed to other transport services such as bus or subway, the duration of travel, departure and arrival position is explicitly recorded in these systems. This feature turns bike sharing system into a virtual sensor network that can be used for sensing mobility in the city. Hence, it is expected that most of important events in the city could be detected via monitoring these data.

==

Data Set

Bike-sharing rental process is highly correlated to the environmental and seasonal settings. For instance, weather conditions, precipitation, day of week, season, hour of the day, etc. can affect the rental behaviors. The core data set is related to
the two-year historical log corresponding to years 2011 and 2012 from Capital Bikeshare system, Washington D.C., USA which is publicly available in http://capitalbikeshare.com/system-data (http://capitalbikeshare.com/system-data). We aggregated the data on two hourly and daily basis and then extracted and added the corresponding weather and seasonal information. Weather information are extracted from http://www.freemeteo.com (http://www.freemeteo.com).

==

Associated tasks

```
- Regression:
    Predication of bike rental count hourly or daily based on the environm
ental and seasonal settings.

- Event and Anomaly Detection:
    Count of rented bikes are also correlated to some events in the town w
hich easily are traceable via search engines.
    For instance, query like "2012-10-30 washington d.c." in Google return
s related results to Hurricane Sandy. Some of the important events are
    identified in [1]. Therefore the data can be used for validation of an
omaly or event detection algorithms as well.
```

===

Files

```
- Readme.txt
- hour.csv : bike sharing counts aggregated on hourly basis. Records: 1737
9 hours
- day.csv - bike sharing counts aggregated on daily basis. Records: 731 da
ys
```

===

Dataset characteristics

Both hour.csv and day.csv have the following fields, except hr which is not available in day.csv

```
- instant: record index
- dteday : date
- season : season (1:springer, 2:summer, 3:fall, 4:winter)
- yr : year (0: 2011, 1:2012)
- mnth : month ( 1 to 12)
- hr : hour (0 to 23)
- holiday : weather day is holiday or not (extracted from http://dchr.dc.g
ov/page/holiday-schedule)
- weekday : day of the week
- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
+ weathersit :
    - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
    - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light R
ain + Scattered clouds
    - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp : Normalized temperature in Celsius. The values are divided to 41
 (max)
- atemp: Normalized feeling temperature in Celsius. The values are divided
to 50 (max)
- hum: Normalized humidity. The values are divided to 100 (max)
- windspeed: Normalized wind speed. The values are divided to 67 (max)
- casual: count of casual users
- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered
```

Goal: To build a prediction model for the hourly utilization “cnt”

In [2]:

```
import pandas as pd

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
# Increase the default plot size and set the color scheme
plt.rcParams['figure.figsize'] = 8, 5
plt.rcParams['image.cmap'] = 'viridis'

from scipy import stats
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor,RandomForestClassifier, AdaBo
ostRegressor,GradientBoostingRegressor, ExtraTreesRegressor, BaggingRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold, Strati
fiedKFold, train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import ElasticNet, Lasso,LassoCV, Ridge
from sklearn.tree import DecisionTreeRegressor
import xgboost as xgb
import lightgbm as lgb


# Metrics
from sklearn.metrics import r2_score, median_absolute_error, mean_absolute_error
from sklearn.metrics import median_absolute_error, mean_squared_error, mean_squa
red_log_error



import warnings
def ignore_warn(*args, **kwargs):
    pass
warnings.warn = ignore_warn #ignore annoying warning (from sklearn and seaborn)
```

In [4]:

```
df_hour = pd.read_csv('./hour.csv')
df_hour.head(10)
```

Out[4]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	a
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	(
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	(
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	(
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	(
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	(
5	6	2011-01-01	1	0	1	5	0	6	0	2	0.24	(
6	7	2011-01-01	1	0	1	6	0	6	0	1	0.22	(
7	8	2011-01-01	1	0	1	7	0	6	0	1	0.20	(
8	9	2011-01-01	1	0	1	8	0	6	0	1	0.24	(
9	10	2011-01-01	1	0	1	9	0	6	0	1	0.32	(

In [5]:

```
# We have around 17300 of samples

df_hour.shape
```

Out[5]:

```
(17379, 17)
```

In [6]:

```
df_hour.describe()
```

Out[6]:

	instant	season	yr	mnth	hr	holiday	
count	17379.0000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17
mean	8690.0000	2.501640	0.502561	6.537775	11.546752	0.028770	
std	5017.0295	1.106918	0.500008	3.438776	6.914405	0.167165	
min	1.0000	1.000000	0.000000	1.000000	0.000000	0.000000	
25%	4345.5000	2.000000	0.000000	4.000000	6.000000	0.000000	
50%	8690.0000	3.000000	1.000000	7.000000	12.000000	0.000000	
75%	13034.5000	3.000000	1.000000	10.000000	18.000000	0.000000	
max	17379.0000	4.000000	1.000000	12.000000	23.000000	1.000000	

In [7]:

```
# No missing values. Therefore we dont need any kind of data imputation.

df_hour.isnull().sum()
```

Out[7]:

```
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
```

In [8]:

```
# The number of unique 'instant' values is equal to the number of samples, there
fore, I think that this
# feature can be neglected for further analysis

len(df_hour.instant.unique())
```

Out[8]:

```
17379
```

EDA

For the sake of simplicity lets ommit time feature.

In [9]:

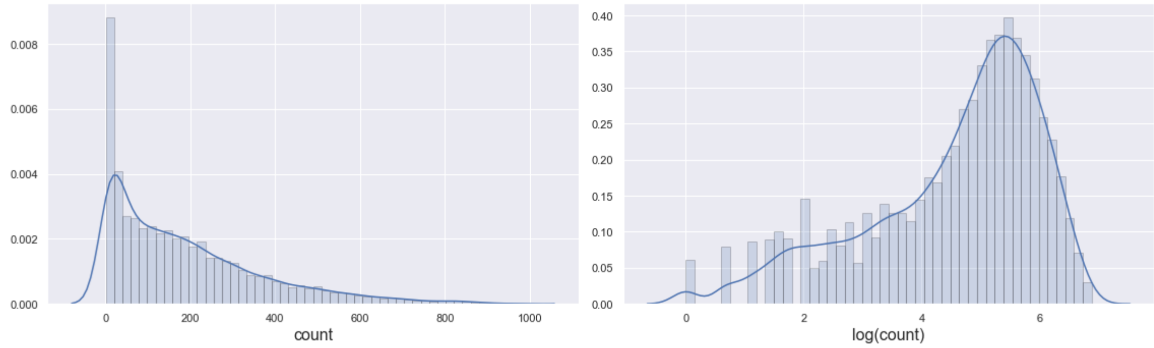
```
# Let us drop the 'instant' column, dont really useful for regression purposes

df = df_hour.drop(['instant','dteday'], axis=1)
df.head(5)
```

Out[9]:

	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	win
0	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	
1	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	
2	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	
3	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	
4	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	

In [10]:

```
# Lets check underlying distribution of 'cnt' variable. Looks more like Poisson
 distribution.
# For some ML algorithms like OLS and it is derivatives (Ridge, Lasso) we would
 like to have
# normally distributed target vector (counts). Hence we would need to log transf
orm it.

# Distribution of cnt
warnings.filterwarnings('ignore')
hist_kws={'histtype': 'bar', 'edgecolor':'black', 'alpha': 0.2}

fig,ax = plt.subplots(nrows=1, ncols=2, figsize=(16,5))
sns.distplot(df['cnt'], ax=ax[0], hist_kws=hist_kws)
sns.distplot(np.log(df['cnt']), ax=ax[1], hist_kws=hist_kws)
ax[0].set_xlabel('count', fontsize=16)
ax[1].set_xlabel('log(count)', fontsize=16)
plt.tight_layout()
plt.show()
```



In the above plot, we see that the variable `cnt` is prominently skewed right which reminds of POisson distribution. The log tranform looks normally distributed. Lets conduct `Shapiro-Wilk` test in order to prove it.

In [11]:

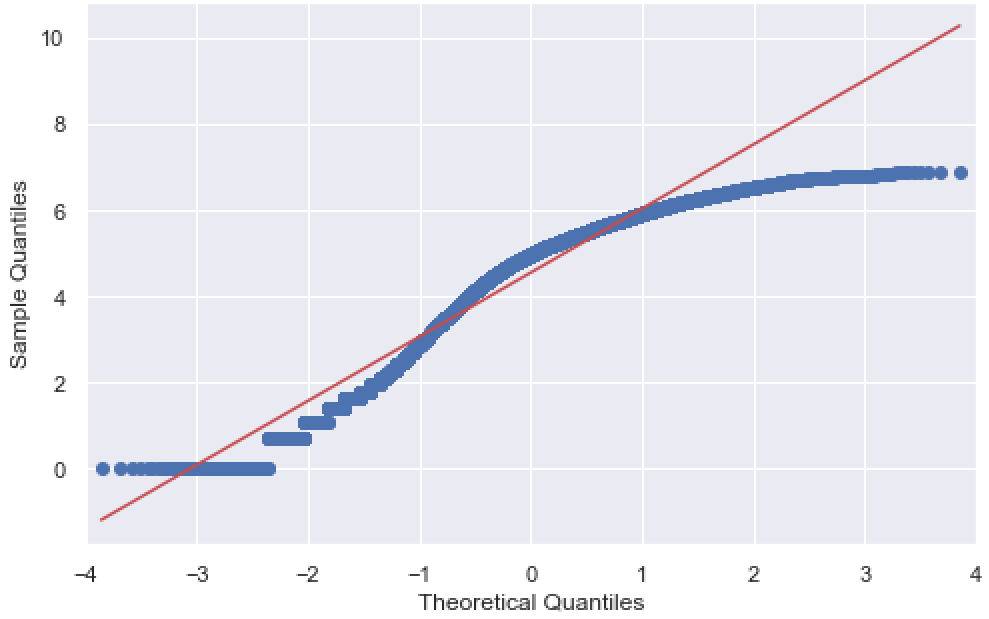
```
stat, p = stats.shapiro(np.log(df['cnt']))
print('Statistics=%.3f, p=%.3f' % (stat, p))

# interpret
alpha = 0.05

if p > alpha:
        print('Sample looks Gaussian (fail to reject H0)')
else:
        print('Sample does not look Gaussian (reject H0)')
```

```
Statistics=0.920, p=0.000
Sample does not look Gaussian (reject H0)
```

The target vector seems to be not normal, however let us conduct one more test and build Q-Q plot.

In [12]:

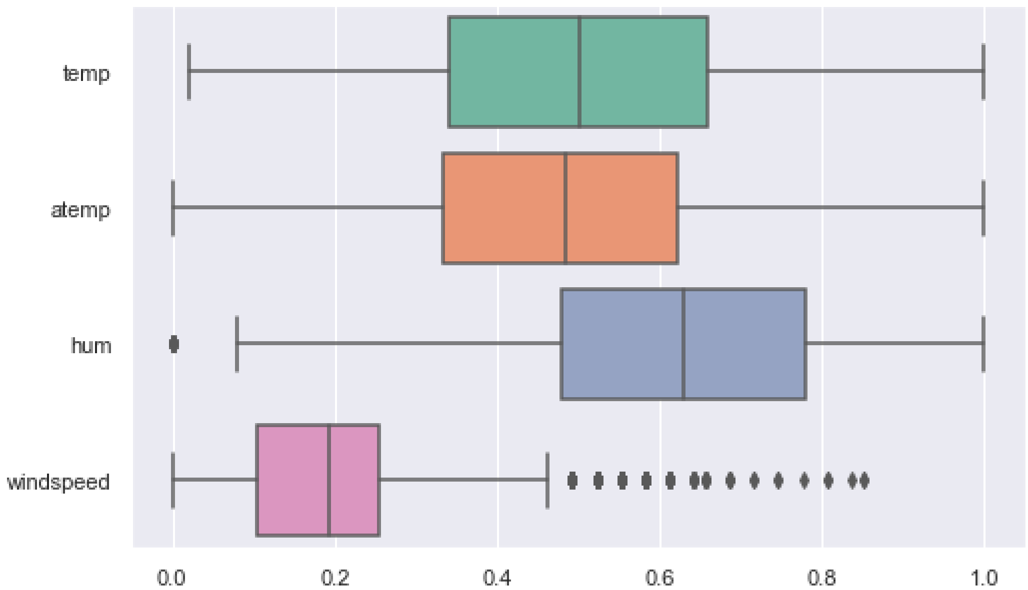
```
import statsmodels.api as sm
import pylab

sm.qqplot(np.log(df['cnt']), line='s')

pylab.show()
```



So our our first assumption is proved that even log-transform of the target vector is not normally distributed, therefore we would expect the linear regressions methods to fail.

In [13]:

```
# Outliers. Let us check numerical features for outliers, we will do that with b
ox plots

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

ax = sns.boxplot(data=df[features], orient="h", palette="Set2")
```



The one can see quite some outliers in `windspeed` feature and a few in `humidity`. We sould check maximum values and get rid of outliers.

In [14]:

```
df.sort_values(by='windspeed', ascending=False).head(20)
```

Out[14]:

	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum
4315	3	0	7	17	0	0	0	3	0.80	0.7424	0.49
4316	3	0	7	18	0	0	0	3	0.80	0.7424	0.49
5635	3	0	8	17	0	6	0	3	0.64	0.5758	0.89
9956	1	1	2	21	0	5	1	1	0.42	0.4242	0.35
1259	1	0	2	15	0	5	1	1	0.46	0.4545	0.41
1017	1	0	2	1	0	2	1	1	0.30	0.2424	0.42
1261	1	0	2	17	0	5	1	1	0.32	0.2727	0.49
1125	1	0	2	15	0	6	0	1	0.44	0.4394	0.16
9653	1	1	2	4	0	0	0	2	0.10	0.0455	0.46
11024	2	1	4	12	0	1	1	1	0.54	0.5152	0.28
9652	1	1	2	3	0	0	0	2	0.10	0.0455	0.46
10690	2	1	3	13	0	1	1	1	0.48	0.4697	0.29
9958	1	1	2	23	0	5	1	1	0.38	0.3939	0.37
10261	1	1	3	15	0	4	1	1	0.64	0.6212	0.38
1018	1	0	2	2	0	2	1	1	0.28	0.2273	0.41
1007	1	0	2	15	0	1	1	1	0.56	0.5303	0.21
1014	1	0	2	22	0	1	1	1	0.34	0.2879	0.46
10263	1	1	3	17	0	4	1	1	0.62	0.6212	0.38
17153	1	1	12	12	0	6	0	1	0.30	0.2576	0.36
1119	1	0	2	9	0	6	0	1	0.40	0.4091	0.16

The top-2 values belong to the same day, 5 and 6 am. Maybe there was some extraordinary storm or smth, don't really think that we should clear this column.

In [15]:

```
df.sort_values(by='hum', ascending=True).head(10)
```

Out[15]:

	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum
1565	1	0	3	16	0	4	1	3	0.42	0.4242	0.0
1570	1	0	3	21	0	4	1	3	0.36	0.3485	0.0
1568	1	0	3	19	0	4	1	3	0.44	0.4394	0.0
1567	1	0	3	18	0	4	1	3	0.44	0.4394	0.0
1566	1	0	3	17	0	4	1	2	0.44	0.4394	0.0
1564	1	0	3	15	0	4	1	3	0.44	0.4394	0.0
1563	1	0	3	14	0	4	1	3	0.44	0.4394	0.0
1562	1	0	3	13	0	4	1	3	0.42	0.4242	0.0
1561	1	0	3	12	0	4	1	3	0.42	0.4242	0.0
1571	1	0	3	22	0	4	1	2	0.34	0.3333	0.0

Having `hum` equal to 0 is quite an outlier because the lowest humidity is found in Antarctica where it is so cold all the moisture has frozen out of the air as frost. Therefore we need to count how many such samples do we have.

In [16]:

```
df[df['hum'] == 0.0].count()
```

Out[16]:

```
season        22
yr            22
mnth          22
hr            22
holiday       22
weekday       22
workingday    22
weathersit    22
temp          22
atemp         22
hum           22
windspeed     22
casual        22
registered    22
cnt           22
dtype: int64
```

So we have only 22 samples out of 18000. We could simply delete these, because some ML models are very sensitive to outliers (i.e SVM) on the other hand if we would have such samples in a test set, deleting such outlier means to lose generalization capacity.

In [17]:

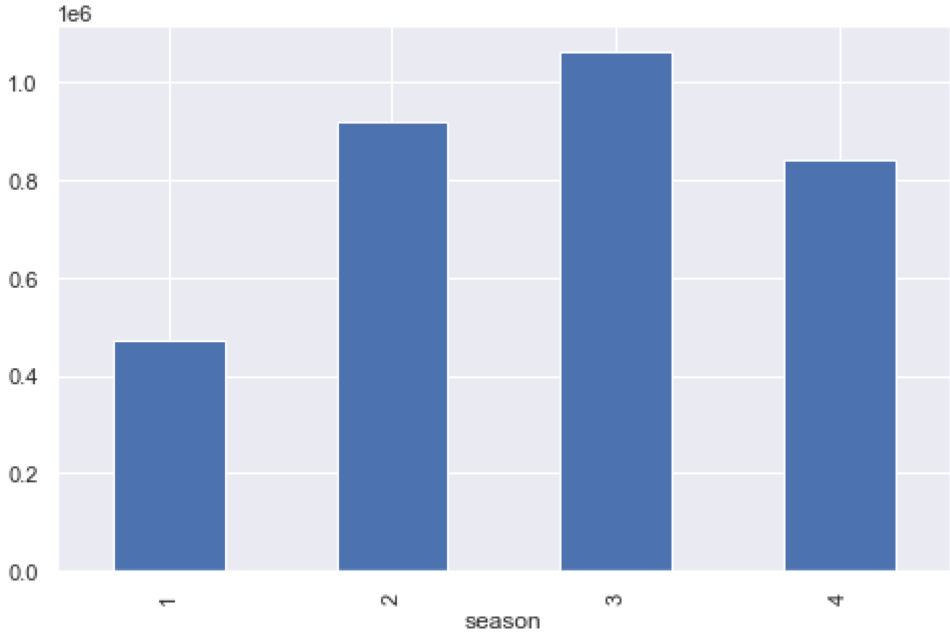
```
# In some cases, we may want to plot a scatterplot matrix such as the one shown
 below.
# Its diagonal contains the distributions of the corresponding variables, and
# the scatter plots for each pair of variables fill the rest of the matrix.


%config InlineBackend.figure_format = 'png'
sns.pairplot(df);
```

In [18]:

```
df.groupby(['season'])['cnt'].sum().plot(kind='bar')
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe326573d0>
```



In [19]:

```
df.groupby(['season'])['cnt'].sum()
```

Out[19]:

```
season
1     471348
2     918589
3    1061129
4     841613
Name: cnt, dtype: int64
```

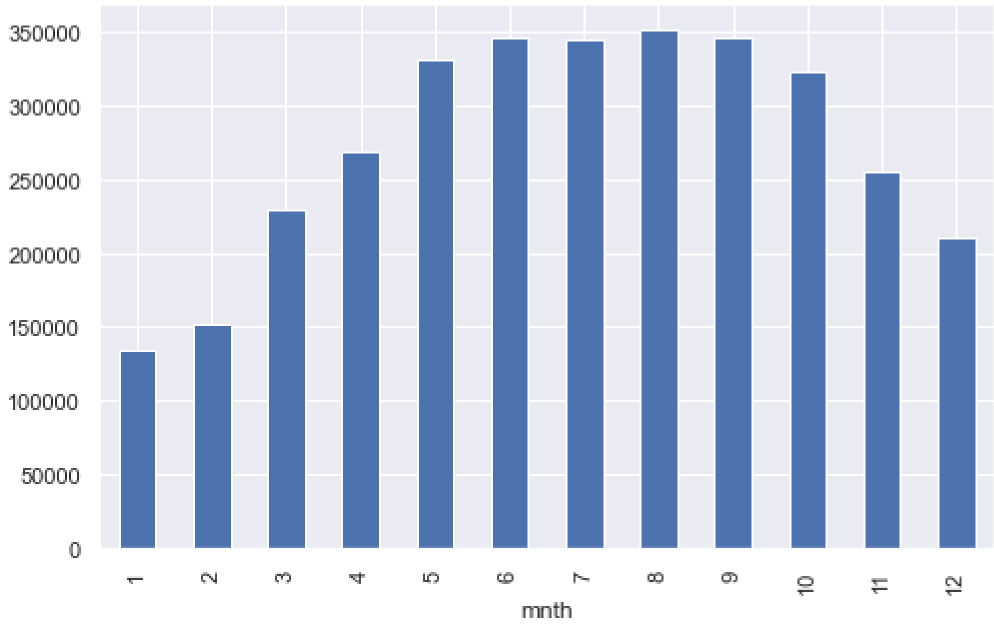
It is clear that the *summer* months have the highest count, although *autumn* monhts are not that far away

In [23]:

```
# Distribution over months is oppoite to what we ve observed above

df.groupby(['mnth'])['cnt'].sum().plot(kind='bar')
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe357a9310>
```



If we plot the `cnt` values w.r.t months, we see that three summer months have bigger values (expected), than any other months.

In [24]:

```
# Lets count manually count of rents in three 'summer' monts

df[df_hour['mnth'].apply(lambda x: (x == 6)|(x == 7)|(x == 8))]['cnt'].sum()
```

Out[24]:

```
1042484
```

In [25]:

```
# Lets count manually anount of rents in three 'autumn' monts

df[df['mnth'].apply(lambda x: (x == 9)|(x == 10)|(x == 11))]['cnt'].sum()
```

Out[25]:

```
923174
```

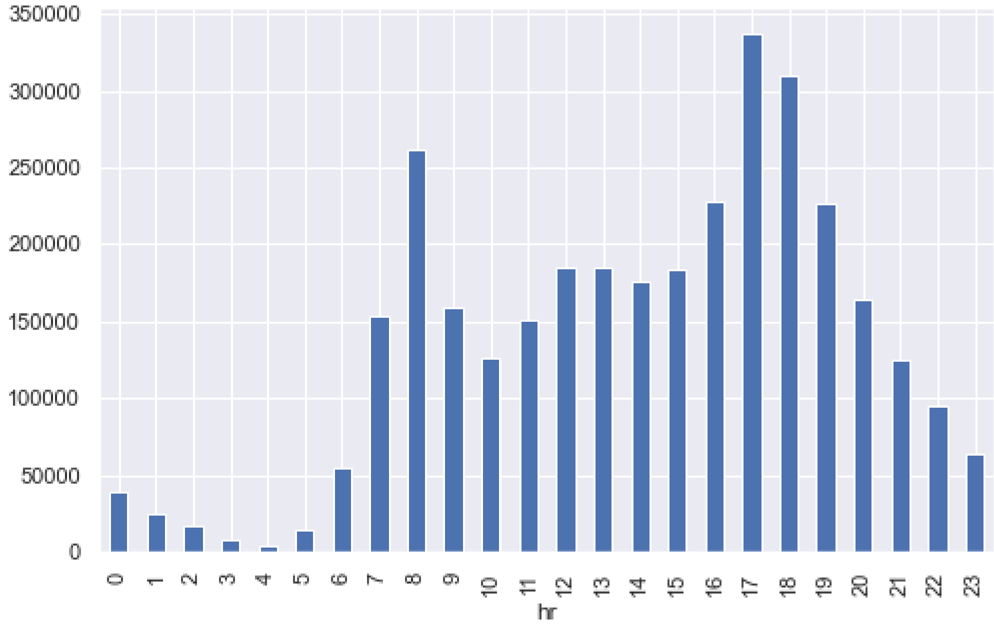
Our assumption is correct

In [26]:

```
# Now let see the distribution of rental bikes w r t hours

df.groupby(['hr'])['cnt'].sum().plot(kind='bar')
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe358c1820>
```



The pattern is pretty clear, very low values at night, than bikes rental starts to grow more or less linearly. Two spikes which can be observed at 8 am and 5 pm, i believe correspond to the time when people go to/from work.

Next is to check how cnt feature relates to weather. Weathersit :

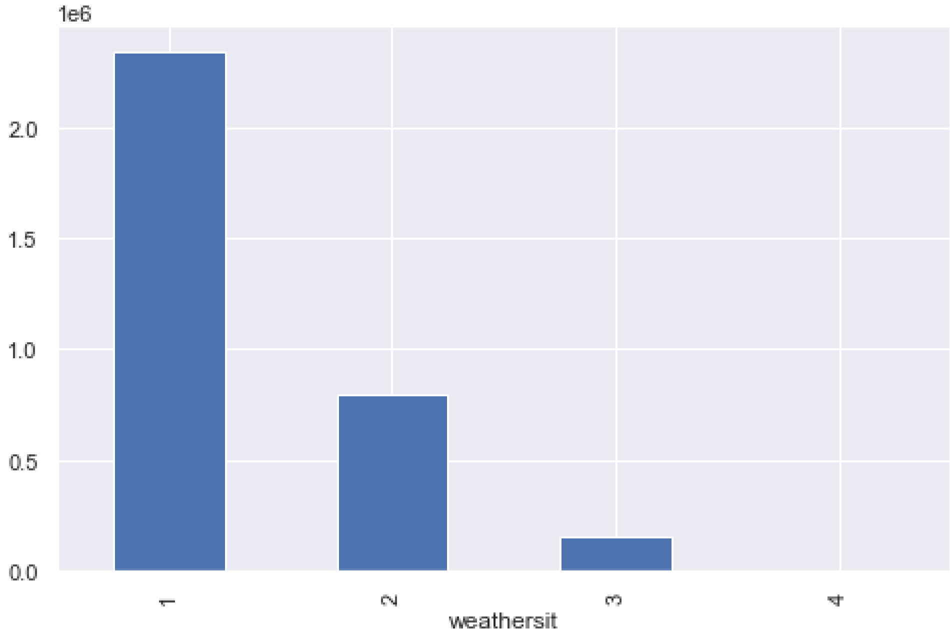
- 1: Clear, Few clouds, Partly cloudy, Partly cloudy
- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
- 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

In [27]:

```
df.groupby(['weathersit'])['cnt'].sum().plot(kind='bar')
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe358fae50>
```



As we can see the majority of the people drive bike only when the weather conditions are good.

In [28]:

```
# What about holydays and working days

df.groupby(['holiday'])['cnt'].sum()
```

Out[28]:

```
holiday
0    3214244
1      78435
Name: cnt, dtype: int64
```

In [29]:

```
df.groupby(['workingday'])['cnt'].sum()
```

Out[29]:

```
workingday
0    1000269
1    2292410
Name: cnt, dtype: int64
```

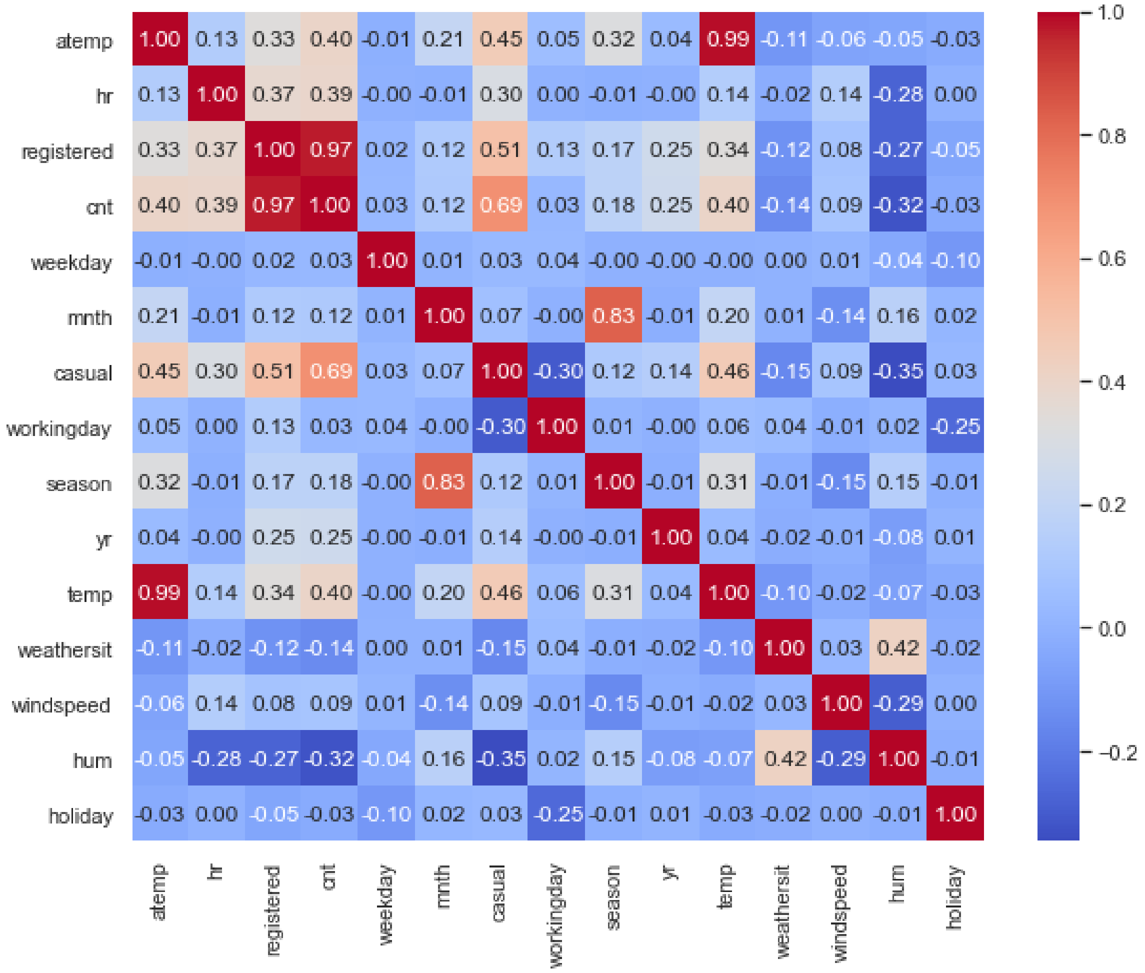
As we would expect people tend to do rent bikes in working days, to go to/from work. This conncusion is supported by hour observation.

In [30]:

```
#Lets plot simple correlation matrix

plt.figure(figsize=(10, 8))
numerical = list(set(df.columns))
corr_matrix = df[numerical].corr()
sns.heatmap(corr_matrix,annot=True,fmt = ".2f", cmap = "coolwarm")
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe359269d0>
```



The one can see that `cnt` feature is highly correlated with `hrs` , `casual` , `temperature` and all other features which represent weather conditions. What is surprising is a weak correlation with `weekday` and `working day` . There is an almost linear relation between `cnt` and `registered` , since `cnt` is consist of `registered` and `casual` , we can also drop these features as they will not help us model demand from single user behavior and work only with the `total count` . Basically, if we do not drop them, we will introduce leakage our linear models like `Ridge` and `Lasso` outperform all other and show minimum *MAE* and *RMSE*.