London Bike Hire Prediction Modelling

- Name name: Raye Yoo
- Project completed date/time: Friday 21 July 2023

Business Problem

Transport for London ("TFL") is currently facing issues to predict bike-hire demand post-pandemic in London. Londoners are going back to the offices and tourists are coming back to London significantly. To solve this problem, I will find out the most impactful features of hiring bikes in London through multiple regression analysis to predict bike-hire demands for the next years.

1. Data Scrub

1-1. Import Data

In [1]:

```
import pandas as pd
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
plt.style.use('ggplot')

from statsmodels.formula.api import ols
from sklearn import preprocessing
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error

data = pd.read_csv('data/london_merged.csv')
```

1-2. Cleaning & Exploring Data

1-2-1. Cleaning Data

In [2]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17414 entries, 0 to 17413
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   timestamp     17414 non-null  object
 1   cnt           17414 non-null  int64
 2   t1            17414 non-null  float64
 3   t2            17414 non-null  float64
 4   hum           17414 non-null  float64
 5   wind_speed    17414 non-null  float64
 6   weather_code  17414 non-null  float64
 7   is_holiday    17414 non-null  float64
 8   is_weekend    17414 non-null  float64
 9   season        17414 non-null  float64
dtypes: float64(8), int64(1), object(1)
memory usage: 1.3+ MB
```

In [3]:
```
data.head()
```

Out[3]:

	timestamp	cnt	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season
0	2015-01-04 00:00:00	182	3.0	2.0	93.0	6.0	3.0	0.0	1.0	3.0
1	2015-01-04 01:00:00	138	3.0	2.5	93.0	5.0	1.0	0.0	1.0	3.0
2	2015-01-04 02:00:00	134	2.5	2.5	96.5	0.0	1.0	0.0	1.0	3.0
3	2015-01-04 03:00:00	72	2.0	2.0	100.0	0.0	1.0	0.0	1.0	3.0
4	2015-01-04 04:00:00	47	2.0	0.0	93.0	6.5	1.0	0.0	1.0	3.0

Metadata:

- timestamp - timestamp field for grouping the data by hours
- cnt - the count of a new bike shares
- t1 - Observed temperature in Celsius
- t2 - "feels like" temperature in Celsius
- hum - humidity in percentage
- wind_speed - wind speed in km/h
- weather_code - category of the weather
- is_holiday - boolean field - 1 holidays / 0 non holidays
- is_weekend - boolean field - 1 weekends / 0 weekdays
- season - category field meteorological seasons: 0-spring; 1-summer; 2-fall; 3-winter

weather_code category description:

1 = Clear ; mostly clear but have some values with haze/fog/patches of fog/ fog in vicinity 2 = scattered clouds / few clouds 3 = Broken clouds 4 = Cloudy 7 = Rain/ light Rain shower/ Light rain 10 = rain with thunderstorm 26 = snowfall 94 = Freezing Fog

Source: Kaggle (https://www.kaggle.com/datasets/hmavrodiev/london-bike-sharing-dataset (https://www.kaggle.com/datasets/hmavrodiev/london-bike-sharing-dataset))

In [4]:
```
data.describe()
```

Out[4]:

	cnt	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season
count	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000
mean	1143.101642	12.468091	11.520836	72.324954	15.913063	2.722752	0.022051	0.285403	1.492075
std	1085.108068	5.571818	6.615145	14.313186	7.894570	2.341163	0.146854	0.451619	1.118911
min	0.000000	-1.500000	-6.000000	20.500000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	257.000000	8.000000	6.000000	63.000000	10.000000	1.000000	0.000000	0.000000	0.000000
50%	844.000000	12.500000	12.500000	74.500000	15.000000	2.000000	0.000000	0.000000	1.000000
75%	1671.750000	16.000000	16.000000	83.000000	20.500000	3.000000	0.000000	1.000000	2.000000
max	7860.000000	34.000000	34.000000	100.000000	56.500000	26.000000	1.000000	1.000000	3.000000

In [5]:
```
#make the date & time column to date time format
data['timestamp'] = pd.to_datetime(data['timestamp'], infer_datetime_format=True)
```

In [6]:
```
#create separated columns for time, months (1=jan, 2=feb...), and years
data['date'] = pd.to_datetime(data['timestamp']).dt.date
data['time'] = pd.to_datetime(data['timestamp']).dt.time
data['month'] = pd.to_datetime(data['date']).dt.month
data['year'] = pd.to_datetime(data['date']).dt.year
```

In [7]:
```
data.head()
```

Out[7]:

	timestamp	cnt	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season	date	time	month	year
0	2015-01-04 00:00:00	182	3.0	2.0	93.0	6.0	3.0	0.0	1.0	3.0	2015-01-04	00:00:00	1	2015
1	2015-01-04 01:00:00	138	3.0	2.5	93.0	5.0	1.0	0.0	1.0	3.0	2015-01-04	01:00:00	1	2015
2	2015-01-04 02:00:00	134	2.5	2.5	96.5	0.0	1.0	0.0	1.0	3.0	2015-01-04	02:00:00	1	2015
3	2015-01-04 03:00:00	72	2.0	2.0	100.0	0.0	1.0	0.0	1.0	3.0	2015-01-04	03:00:00	1	2015
4	2015-01-04 04:00:00	47	2.0	0.0	93.0	6.5	1.0	0.0	1.0	3.0	2015-01-04	04:00:00	1	2015

In [8]:

```
#update the data type to int for time_hour
data['time_hour'] = data.time.astype(str).str[:2].astype(int)
```

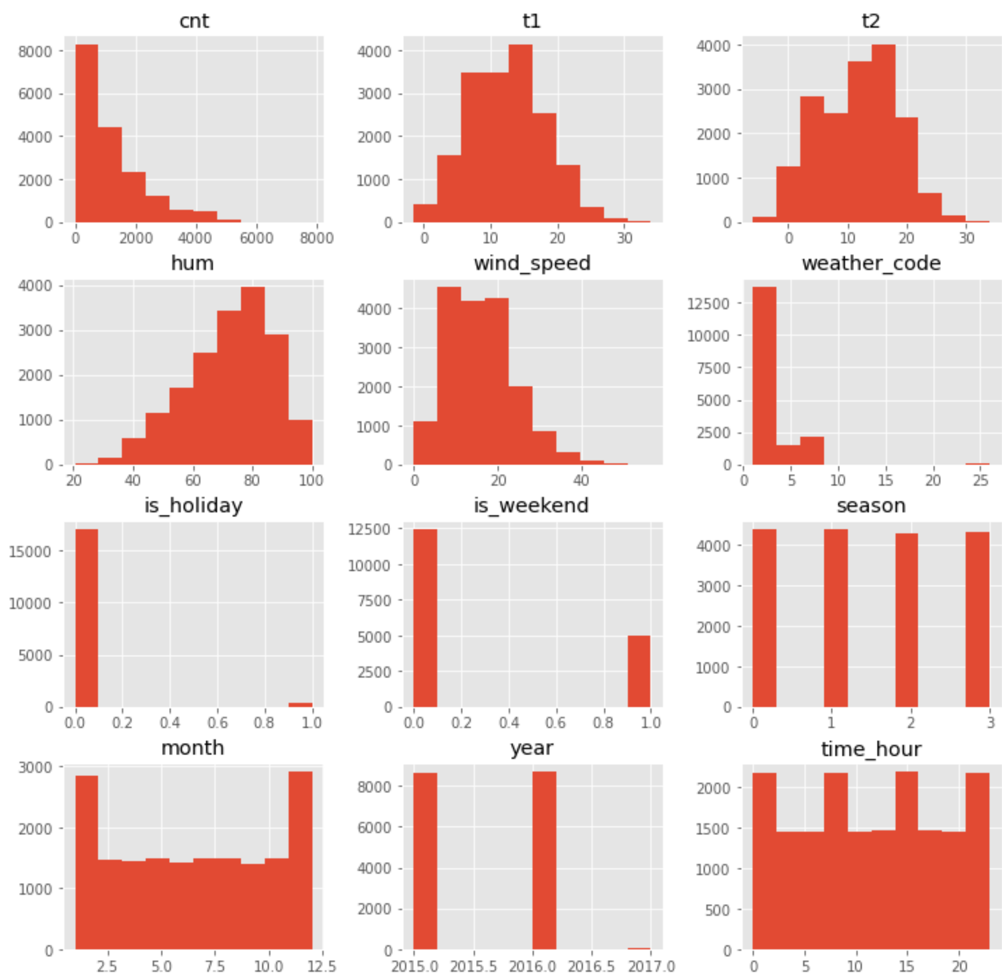
In [9]:

```
#now we drop the timestamp-related variables
data = data.drop(columns=['timestamp','time','date'])
```

1-2-2. Checking Data Distribution

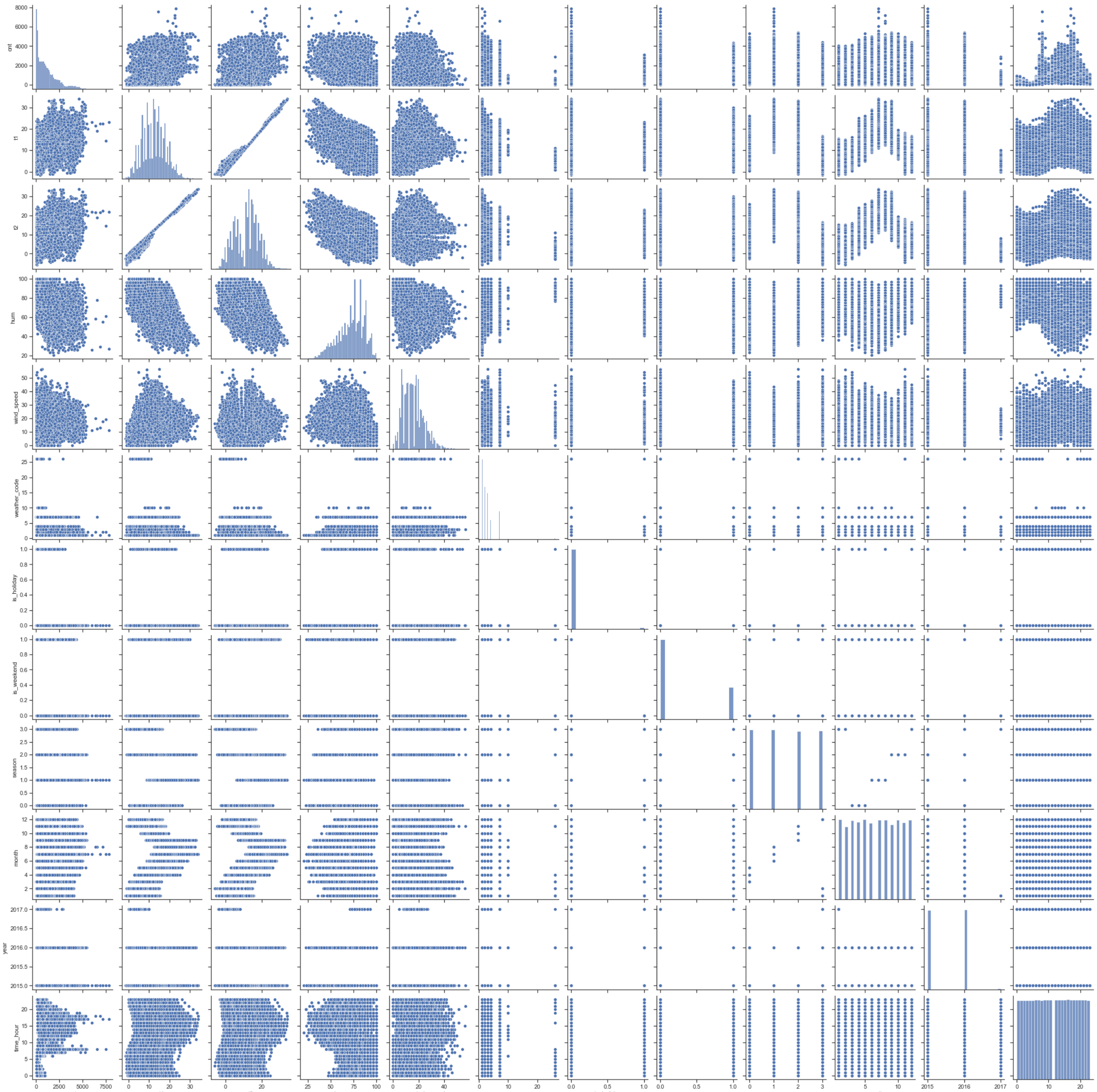
In [10]:

```
data.hist(figsize = (12,12));
#histograms
#count is very skewed - not no negative values since it's adding up by counting, not bell-shaped
#t1, t2, hum and wind_speed are not perfectly, but normally distributed
#the other categorical variables are not shown as normally distributed
```



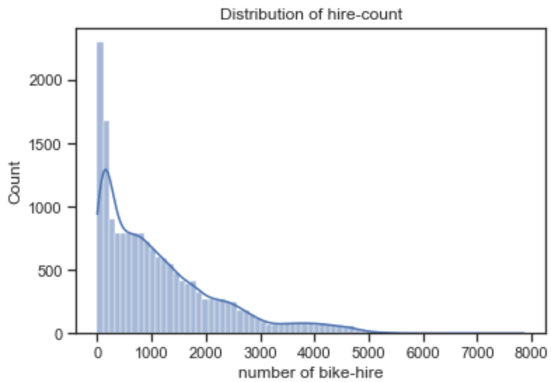
In [11]:

```
sns.set_theme(style="ticks")
sns.pairplot(data);
#Scatter plots
```



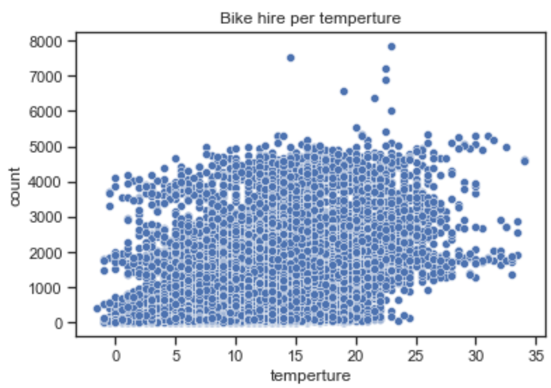
In [12]:

```python
sns.histplot(data['cnt'], kde=True)
plt.title('Distribution of hire-count')
plt.xlabel('number of bike-hire')
plt.savefig("count_hist.png", transparent=True);
#It looks like a skewed normal distribution, but there are no minus values
```



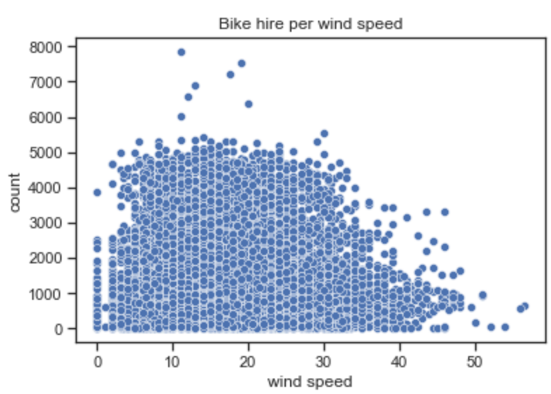
In [13]:

```python
sns.scatterplot(x=data['t1'], y=data['cnt'],data=data)
plt.title('Bike hire per temperture')
plt.xlabel('temperture')
plt.ylabel('count');
#very much slightly linear relationship
```



In [14]:

```python
sns.scatterplot(x=data['wind_speed'], y=data['cnt'],data=data)
plt.title('Bike hire per wind speed')
plt.xlabel('wind speed')
plt.ylabel('count');
#hardly can see any relationship
```

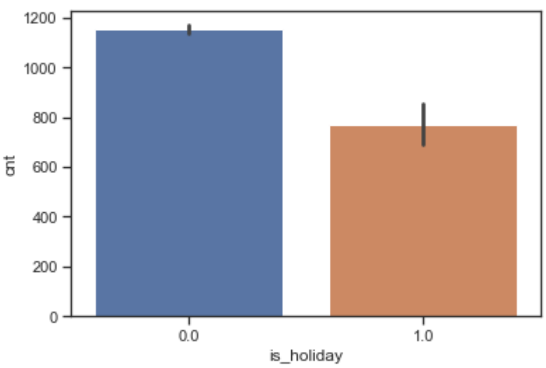


In [15]:

```
sns.scatterplot(x=data['hum'], y=data['cnt'],data=data)
plt.title('Bike hire per humidity %')
plt.xlabel('humidity')
plt.ylabel('count');
#hardly can identify any relation between count and humidity
```

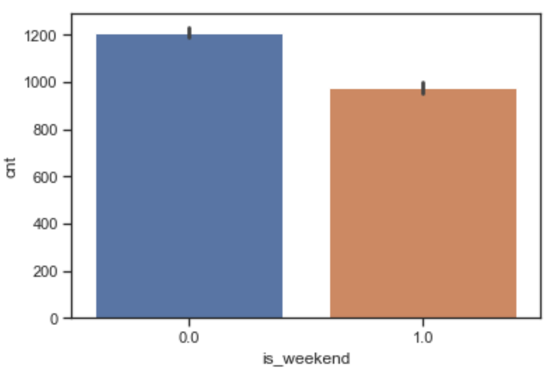
In [16]:

```
sns.barplot(x=data['is_holiday'], y=data['cnt'],data = data)
plt.savefig("count_per_holiday.png", transparent=True);
#Weekdays have more trip counts than holidays
```

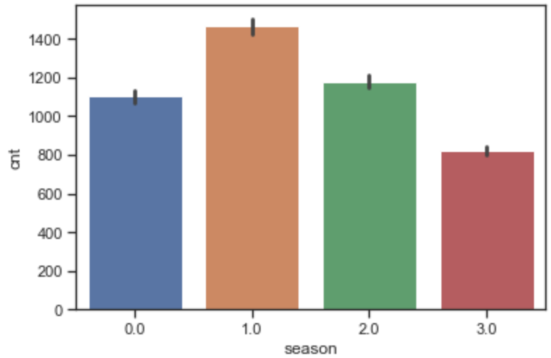


In [17]:

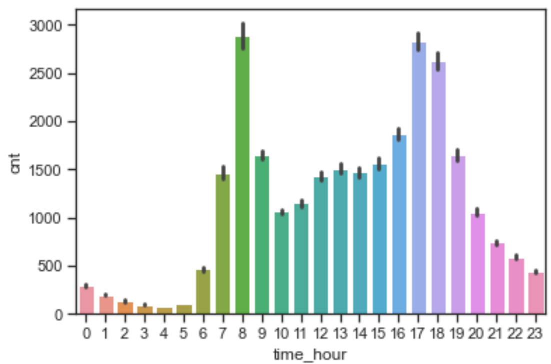
```
sns.barplot(x=data['is_weekend'], y=data['cnt'],data = data)
plt.savefig("count_per_weekend.png", transparent=True);
#Weekdays have more trip counts than weekends
```



In [18]:
```python
sns.barplot(x=data['season'], y=data['cnt'],data = data)
plt.savefig("count__per_seasons.png", transparent=True);
#0.0 spring, 1.0 summer, 2.0 fall, 3.0 winter
#Summer is the most popular time to hire bikes in London
```



In [19]:
```python
sns.barplot(x=data['time_hour'], y=data['cnt'],data = data)
plt.savefig("count_per_time.png", transparent=True);
```



1-2-3. Identifying and removing outliers

In [20]:
```python
df1 = data.copy()
```

In [21]:
```python
#create boxplots for the numerous variables
def boxplot(column):
    sns.boxplot(x=df1[f"{column}"], data=df1)
    plt.title(f"Boxplot of {column}")
    plt.show()
```

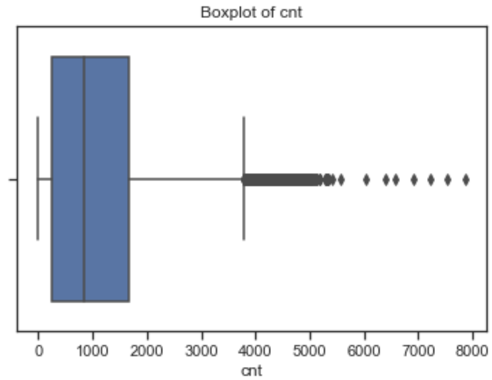
In [22]:
```python
df1.head()
```

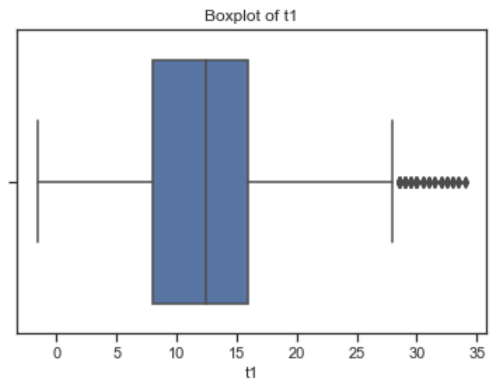
Out[22]:

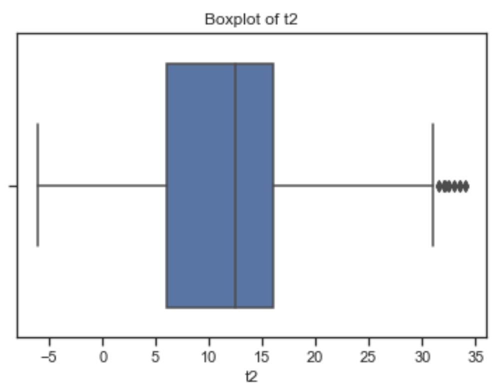
	cnt	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season	month	year	time_hour
0	182	3.0	2.0	93.0	6.0	3.0	0.0	1.0	3.0	1	2015	0
1	138	3.0	2.5	93.0	5.0	1.0	0.0	1.0	3.0	1	2015	1
2	134	2.5	2.5	96.5	0.0	1.0	0.0	1.0	3.0	1	2015	2
3	72	2.0	2.0	100.0	0.0	1.0	0.0	1.0	3.0	1	2015	3
4	47	2.0	0.0	93.0	6.5	1.0	0.0	1.0	3.0	1	2015	4

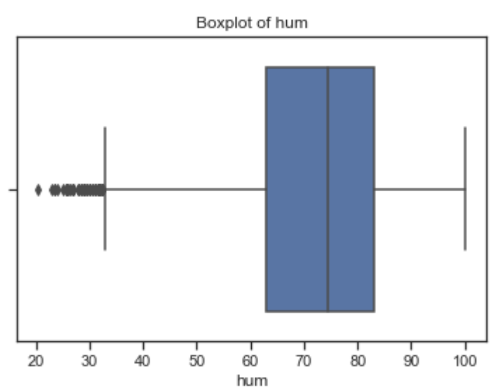
In [23]:

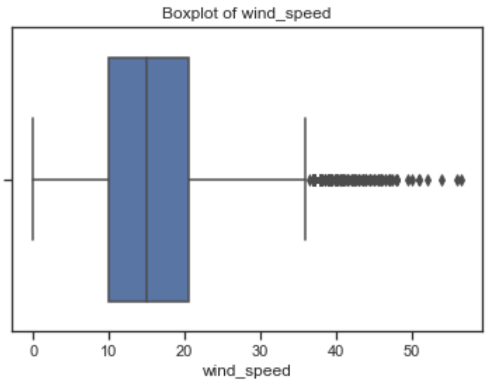
```
boxplot('cnt')
boxplot('t1')
boxplot('t2')
boxplot('hum')
boxplot('wind_speed')
```











In [24]:

```
#remove the outliers
for x in ['cnt']:
    q75,q25 = np.percentile(df1.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    df1.loc[df1[x] < min,x] = np.nan
    df1.loc[df1[x] > max,x] = np.nan
```

In [25]:

```
for x in ['t1']:
    q75,q25 = np.percentile(df1.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    df1.loc[df1[x] < min,x] = np.nan
    df1.loc[df1[x] > max,x] = np.nan
```

In [26]:

```
for x in ['t2']:
    q75,q25 = np.percentile(df1.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    df1.loc[df1[x] < min,x] = np.nan
    df1.loc[df1[x] > max,x] = np.nan
```

In [27]:

```
for x in ['hum']:
    q75,q25 = np.percentile(df1.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    df1.loc[df1[x] < min,x] = np.nan
    df1.loc[df1[x] > max,x] = np.nan
```

In [28]:

```
for x in ['wind_speed']:
    q75,q25 = np.percentile(df1.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    df1.loc[df1[x] < min,x] = np.nan
    df1.loc[df1[x] > max,x] = np.nan
```

In [29]:
```
df1.isnull().sum()
```

Out[29]:
```
cnt             675
t1               64
t2               19
hum              71
wind_speed      236
weather_code      0
is_holiday        0
is_weekend        0
season            0
month             0
year              0
time_hour         0
dtype: int64
```

In [30]:
```
#replace the nan values to median values
df1['cnt'] = df1['cnt'].fillna(df1['cnt'].median())
df1['t1'] = df1['t1'].fillna(df1['t1'].median())
df1['t2'] = df1['t2'].fillna(df1['t2'].median())
df1['hum'] = df1['hum'].fillna(df1['hum'].median())
df1['wind_speed'] = df1['wind_speed'].fillna(df1['wind_speed'].median())
```

In [31]:
```
df1.isnull().sum()
```

Out[31]:
```
cnt             0
t1              0
t2              0
hum             0
wind_speed      0
weather_code    0
is_holiday      0
is_weekend      0
season          0
month           0
year            0
time_hour       0
dtype: int64
```

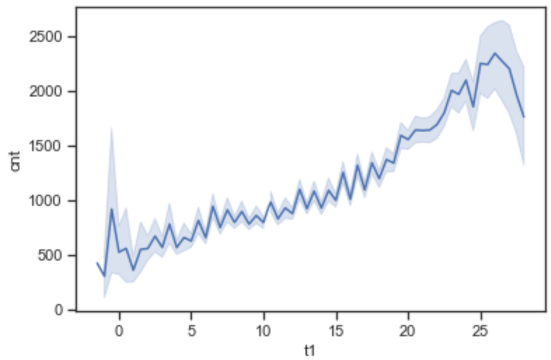
In [32]:
```
df1.describe()
```

Out[32]:

	cnt	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season	mont
count	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.000000	17414.00000
mean	1005.958195	12.401908	11.498928	72.512188	15.563402	2.722752	0.022051	0.285403	1.492075	6.51464
std	871.921136	5.462097	6.578463	14.042904	7.323064	2.341163	0.146854	0.451619	1.118911	3.45250
min	0.000000	-1.500000	-6.000000	33.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.00000
25%	257.000000	8.000000	6.000000	63.000000	10.000000	1.000000	0.000000	0.000000	0.000000	4.00000
50%	798.000000	12.500000	12.500000	75.000000	15.000000	2.000000	0.000000	0.000000	1.000000	7.00000
75%	1493.000000	16.000000	16.000000	83.000000	20.000000	3.000000	0.000000	1.000000	2.000000	10.00000
max	3793.000000	28.000000	31.000000	100.000000	36.000000	26.000000	1.000000	1.000000	3.000000	12.00000

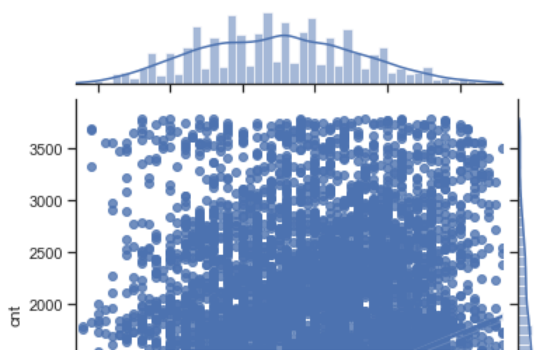
1-2-4. Checking Data Distribution

In [33]:
```python
sns.lineplot( x = "t1",
             y = "cnt",
             data = df1);
```



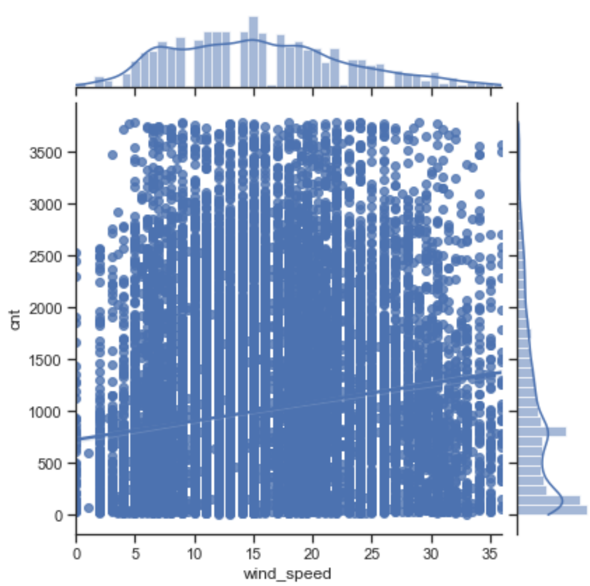
In [34]:
```python
#can see better linear relationship than before removing outliers
sns.jointplot('t1','cnt', data=df1, kind='reg');
```

```
C:\Users\Raye\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the followin
g variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing
other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



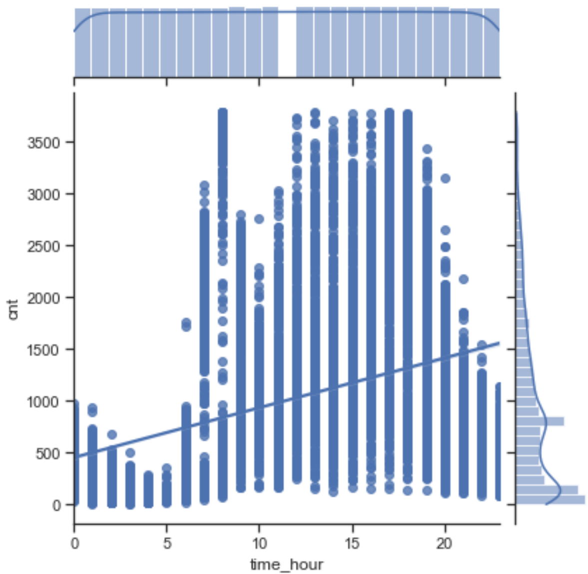
In [35]:
```python
sns.jointplot('wind_speed','cnt', data=df1, kind='reg');
```

```
C:\Users\Raye\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following
variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



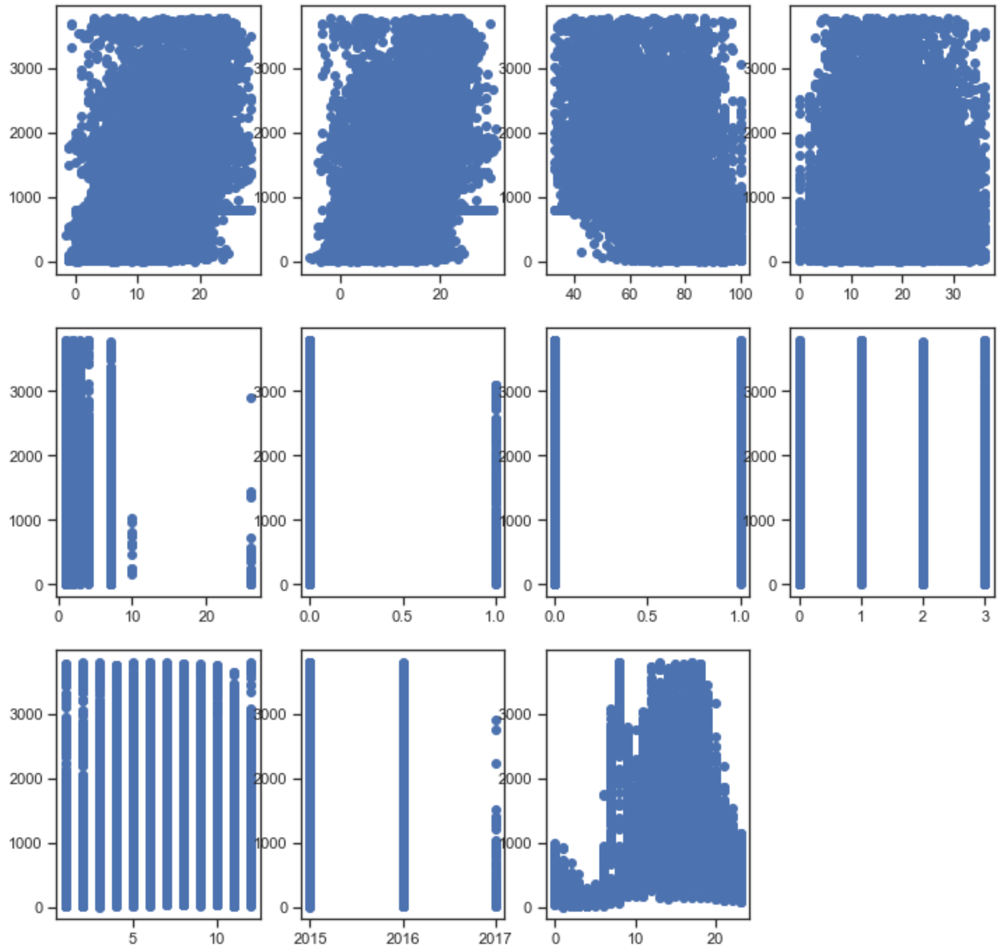
In [36]:
```python
sns.jointplot('time_hour','cnt', data=df1, kind='reg');
```

```
C:\Users\Raye\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



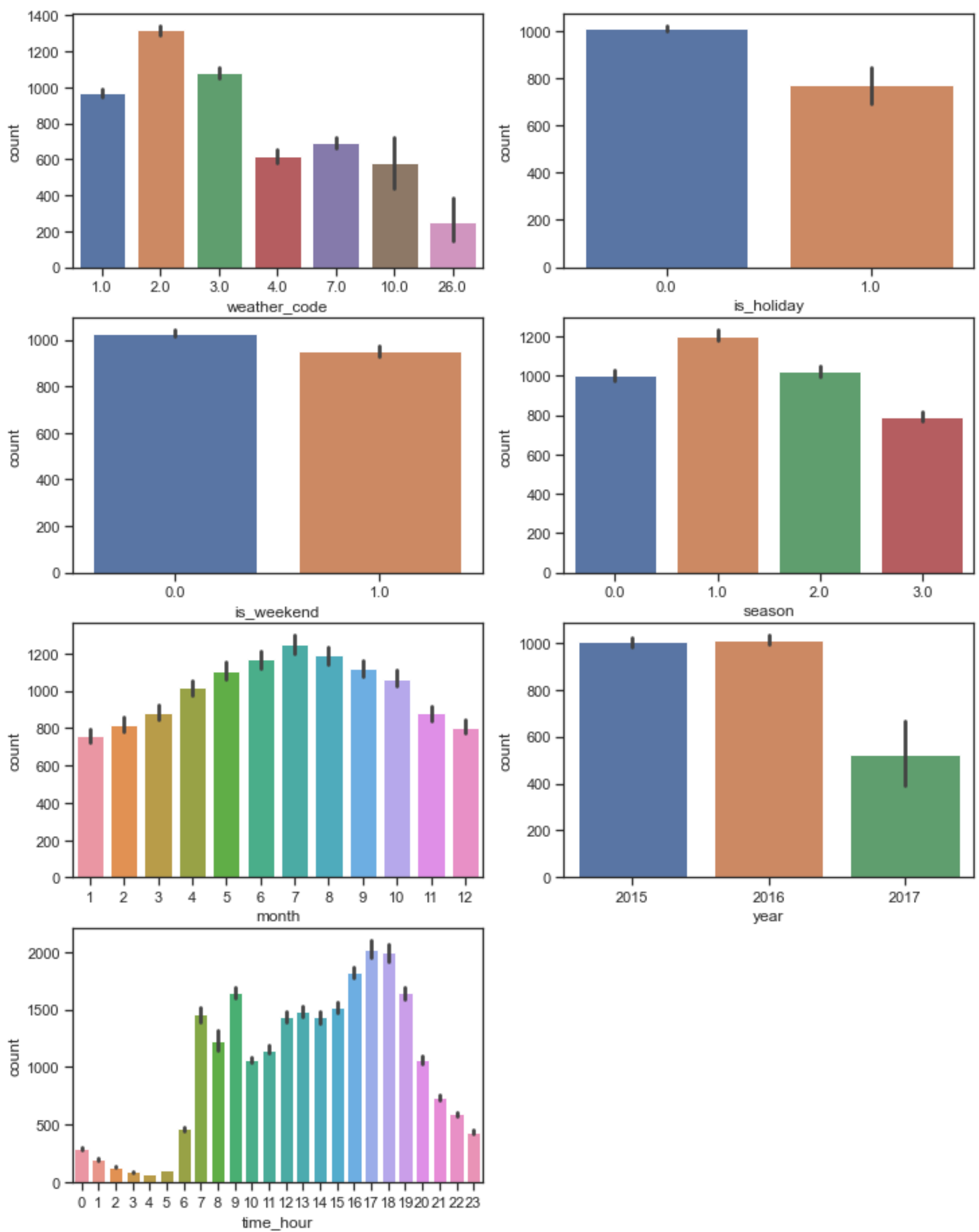
In [37]:

```
#check data distribution after removing outliers

dependent = ['t1','t2','hum','wind_speed',
             'weather_code','is_holiday','is_weekend','season',
             'month','year','time_hour']

plt.figure(figsize=(12,12))

for i in enumerate(dependent):
    plt.subplot(3,4,i[0]+1 )
    plt.scatter(x =i[1], y ='cnt', data= df1);
```



In [38]:

```python
#check distribution with categorical variables

cat = ['weather_code','is_holiday','is_weekend','season',
       'month','year','time_hour']

plt.figure(figsize=(12,16))

for i in enumerate(cat):
    plt.subplot(4, 2, i[0]+1)
    sns.barplot(x=i[1], y=df1['cnt'],data = df1)
    plt.ylabel('count')

plt.show();

#2&3 are the highest weathers for counts, but this is the London's typical weather - just a lot..
#1 = Clear ; mostly clear but have some values with haze/fog/patches of fog/ fog in vicinity
#2 = scattered clouds / few clouds
#3 = Broken clouds
#4 = Cloudy
#7 = Rain/ light Rain shower/ Light rain
#10 = rain with thunderstorm
#26 = snowfall
```



2. Data Modeling

2-1. Baseline Modeling

In [39]:

```
outcome = 'cnt'
prediction = df1.drop('cnt', axis=1)

pred_sum = '+'.join(prediction.columns)
formula = outcome + '~' + pred_sum

model = ols(formula=formula, data=df1).fit()
model.summary()
```

Out[39]:

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.312
Model:	OLS	**Adj. R-squared:**	0.312
Method:	Least Squares	**F-statistic:**	717.7
Date:	Sat, 05 Aug 2023	**Prob (F-statistic):**	0.00
Time:	12:48:05	**Log-Likelihood:**	-1.3936e+05
No. Observations:	17414	**AIC:**	2.787e+05
Df Residuals:	17402	**BIC:**	2.788e+05
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-9.742e+04	2.19e+04	-4.445	0.000	-1.4e+05	-5.45e+04
t1	27.2429	5.240	5.199	0.000	16.972	37.514
t2	1.8101	4.350	0.416	0.677	-6.716	10.336
hum	-20.0419	0.518	-38.712	0.000	-21.057	-19.027
wind_speed	-0.4439	0.838	-0.530	0.596	-2.086	1.198
weather_code	-4.1307	2.568	-1.608	0.108	-9.164	0.903
is_holiday	-153.0192	37.617	-4.068	0.000	-226.753	-79.285
is_weekend	-62.6998	12.215	-5.133	0.000	-86.642	-38.758
season	30.4740	5.553	5.488	0.000	19.589	41.359
month	2.2451	1.902	1.181	0.238	-1.482	5.972
year	49.1833	10.875	4.523	0.000	27.868	70.499
time_hour	32.3123	0.837	38.607	0.000	30.672	33.953

Omnibus:	3671.983	**Durbin-Watson:**	0.862
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	7238.089
Skew:	1.277	**Prob(JB):**	0.00
Kurtosis:	4.859	**Cond. No.**	8.06e+06

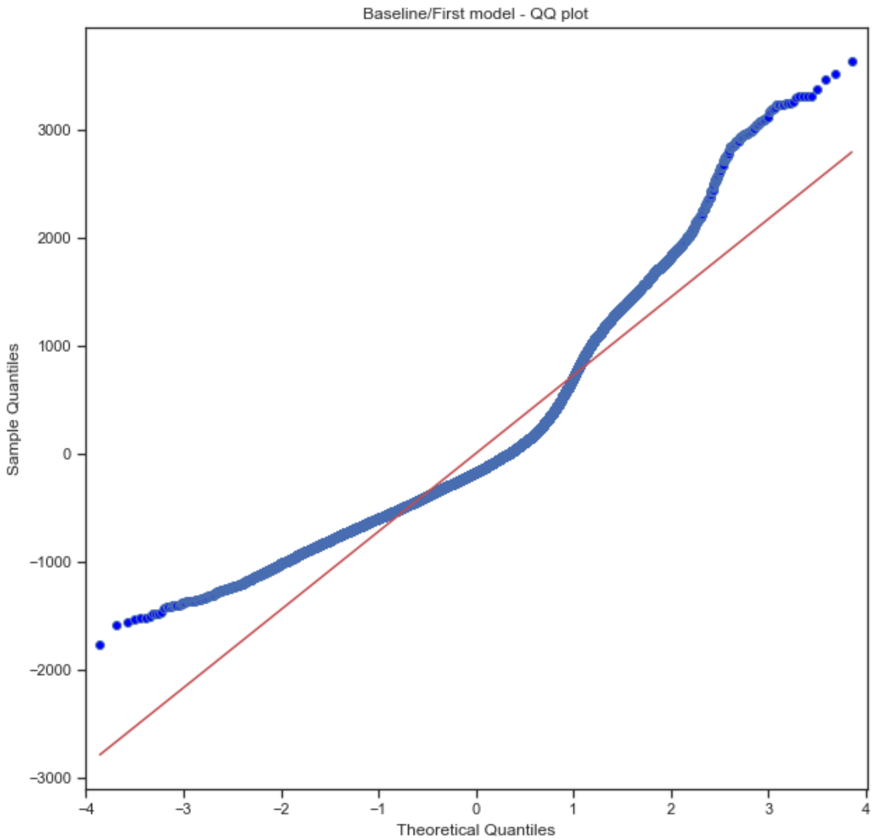
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.06e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

Baseline Model Analysis

- The R squared score is 0.312, yet to not reliable
- t2, wind_speed, weather code and moths show high p-values (> 0.02)
- Skewness and Kurtosis are far from 0

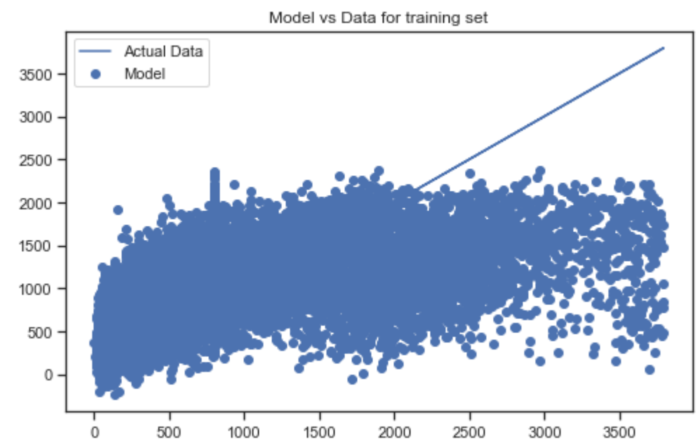
In [40]:

```python
residuals = model.resid

fig, ax = plt.subplots(1,1)
fig.set_figheight(10)
fig.set_figwidth(10)

sm.ProbPlot(residuals).qqplot(line='s',ax=ax)
ax.title.set_text('Baseline/First model - QQ plot');
```



In [41]:

```python
y = df1[['cnt']]
X = df1.drop(['cnt'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.3)

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
train_prediction = linreg.predict(X_train)
test_prediction = linreg.predict(X_test)

plt.figure(figsize=(8,5))
plt.scatter(y_train, train_prediction, label='Model')
plt.plot(y_train, y_train, label='Actual Data')
plt.title('Model vs Data for training set')
plt.legend();
```



2. Second Modeling

2-2-1. Checking Multicollinearity

In [42]:

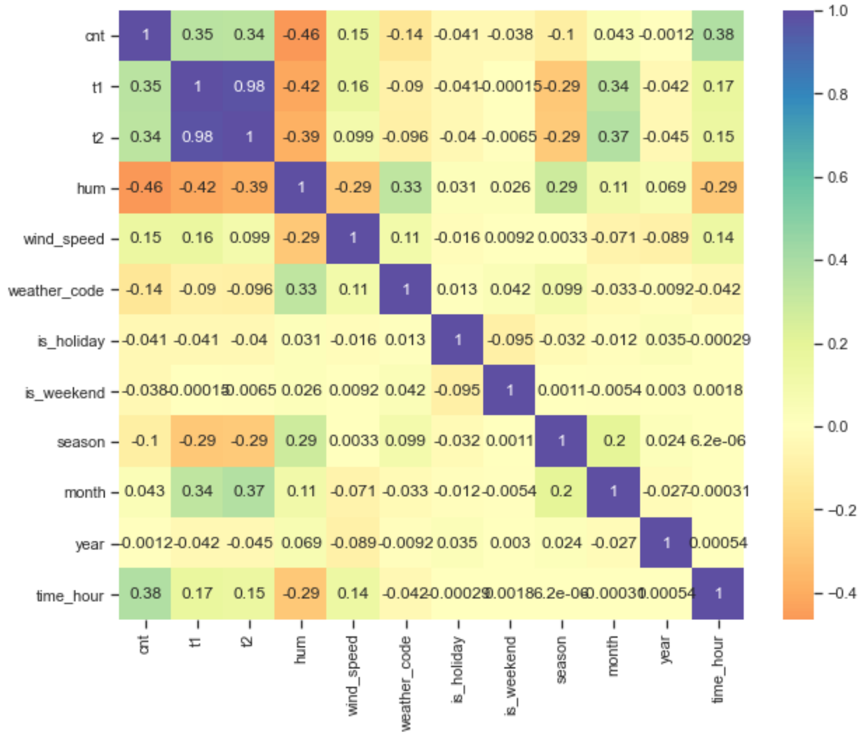
```python
# Check overly correlated variables to produce multicollinearity
corr = df1.corr()
corr
```

Out[42]:

	cnt	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season	month	year	time_h
cnt	1.000000	0.353277	0.337323	-0.464398	0.151575	-0.144278	-0.040719	-0.038341	-0.103975	0.042784	-0.001204	0.381
t1	0.353277	1.000000	0.978943	-0.417577	0.159296	-0.090224	-0.041262	-0.000155	-0.288172	0.335589	-0.042079	0.165
t2	0.337323	0.978943	1.000000	-0.386163	0.098891	-0.096483	-0.039774	-0.006453	-0.286638	0.369407	-0.045262	0.153
hum	-0.464398	-0.417577	-0.386163	1.000000	-0.291789	0.331381	0.030683	0.025515	0.285598	0.113500	0.069301	-0.292
wind_speed	0.151575	0.159296	0.098891	-0.291789	1.000000	0.106355	-0.015505	0.009151	0.003338	-0.071244	-0.089099	0.143
weather_code	-0.144278	-0.090224	-0.096483	0.331381	0.106355	1.000000	0.012939	0.042362	0.098976	-0.033253	-0.009234	-0.041
is_holiday	-0.040719	-0.041262	-0.039774	0.030683	-0.015505	0.012939	1.000000	-0.094898	-0.032488	-0.011511	0.034631	-0.000
is_weekend	-0.038341	-0.000155	-0.006453	0.025515	0.009151	0.042362	-0.094898	1.000000	0.001067	-0.005406	0.003049	0.001
season	-0.103975	-0.288172	-0.286638	0.285598	0.003338	0.098976	-0.032488	0.001067	1.000000	0.203249	0.024400	0.000
month	0.042784	0.335589	0.369407	0.113500	-0.071244	-0.033253	-0.011511	-0.005406	0.203249	1.000000	-0.026547	-0.000
year	-0.001204	-0.042079	-0.045262	0.069301	-0.089099	-0.009234	0.034631	0.003049	0.024400	-0.026547	1.000000	0.000
time_hour	0.381055	0.165728	0.153094	-0.292747	0.143256	-0.041786	-0.000288	0.001803	0.000006	-0.000312	0.000542	1.000

In [43]:

```python
#t1 and t2 are very correlated as expected
plt.figure(figsize=(10,8))
cmap=sns.color_palette("Spectral", as_cmap=True)
sns.heatmap(corr, center=0, annot=True, cmap=cmap);
```



In [44]:

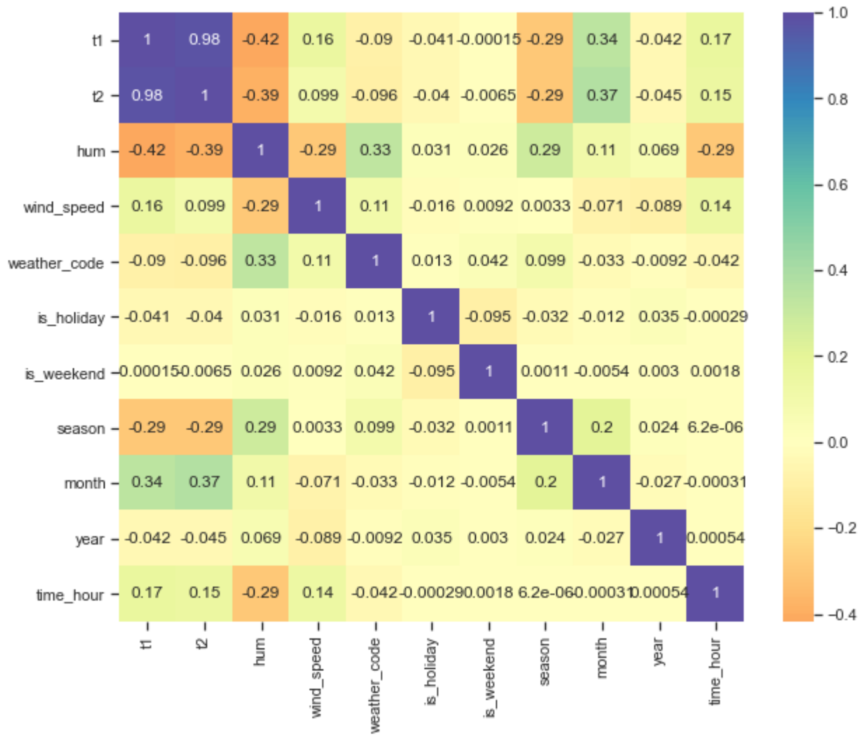
```python
pred_corr = df1[dependent].corr()
pred_corr
```

Out[44]:

	t1	t2	hum	wind_speed	weather_code	is_holiday	is_weekend	season	month	year	time_hour
t1	1.000000	0.978943	-0.417577	0.159296	-0.090224	-0.041262	-0.000155	-0.288172	0.335589	-0.042079	0.165728
t2	0.978943	1.000000	-0.386163	0.098891	-0.096483	-0.039774	-0.006453	-0.286638	0.369407	-0.045262	0.153094
hum	-0.417577	-0.386163	1.000000	-0.291789	0.331381	0.030683	0.025515	0.285598	0.113500	0.069301	-0.292747
wind_speed	0.159296	0.098891	-0.291789	1.000000	0.106355	-0.015505	0.009151	0.003338	-0.071244	-0.089099	0.143256
weather_code	-0.090224	-0.096483	0.331381	0.106355	1.000000	0.012939	0.042362	0.098976	-0.033253	-0.009234	-0.041786
is_holiday	-0.041262	-0.039774	0.030683	-0.015505	0.012939	1.000000	-0.094898	-0.032488	-0.011511	0.034631	-0.000288
is_weekend	-0.000155	-0.006453	0.025515	0.009151	0.042362	-0.094898	1.000000	0.001067	-0.005406	0.003049	0.001803
season	-0.288172	-0.286638	0.285598	0.003338	0.098976	-0.032488	0.001067	1.000000	0.203249	0.024400	0.000006
month	0.335589	0.369407	0.113500	-0.071244	-0.033253	-0.011511	-0.005406	0.203249	1.000000	-0.026547	-0.000312
year	-0.042079	-0.045262	0.069301	-0.089099	-0.009234	0.034631	0.003049	0.024400	-0.026547	1.000000	0.000542
time_hour	0.165728	0.153094	-0.292747	0.143256	-0.041786	-0.000288	0.001803	0.000006	-0.000312	0.000542	1.000000

In [45]:

```python
plt.figure(figsize=(10,8))
sns.heatmap(pred_corr, center=0, annot=True, cmap=cmap);
```



In [46]:

```python
#remove t2 - it also had a high p-value
df2 = df1.drop(columns=['t2'])
```

In [47]:

```python
df2.head()
```

Out[47]:

	cnt	t1	hum	wind_speed	weather_code	is_holiday	is_weekend	season	month	year	time_hour
0	182.0	3.0	93.0	6.0	3.0	0.0	1.0	3.0	1	2015	0
1	138.0	3.0	93.0	5.0	1.0	0.0	1.0	3.0	1	2015	1
2	134.0	2.5	96.5	0.0	1.0	0.0	1.0	3.0	1	2015	2
3	72.0	2.0	100.0	0.0	1.0	0.0	1.0	3.0	1	2015	3
4	47.0	2.0	93.0	6.5	1.0	0.0	1.0	3.0	1	2015	4

2-2-2. Creating Dummy Variables

In [48]:

```python
# make the data type to categories for categorical variables
df2.weather_code = df2.weather_code.astype('category')
df2.is_holiday = df2.is_holiday.astype('category')
df2.is_weekend = df2.is_weekend.astype('category')
df2.season = df2.season.astype('category')
df2.month = df2.month.astype('category')
df2.year = df2.year.astype('category')
df2.time_hour = df2.time_hour.astype('category')
```

In [49]:

```python
#Create dummy variables for the categorical variables
wthr_dummies = pd.get_dummies(df2['weather_code'], prefix='wthr', drop_first=True)
hol_dummies = pd.get_dummies(df2['is_holiday'], prefix='hol', drop_first=True)
wkd_dummies = pd.get_dummies(df2['is_weekend'], prefix='wkd', drop_first=True)
ssn_dummies = pd.get_dummies(df2['season'], prefix='ssn', drop_first=True)
mth_dummies = pd.get_dummies(df2['month'], prefix='mth', drop_first=True)
yr_dummies = pd.get_dummies(df2['year'], prefix='yr', drop_first=True)
time_dummies = pd.get_dummies(df2['time_hour'], prefix='time', drop_first=True)
```

In [50]:

```python
#add dummies to the dataset
df2 = df2.join([wthr_dummies,hol_dummies,wkd_dummies,ssn_dummies,
                mth_dummies,yr_dummies,time_dummies])
```

In [51]:

```
#drop the original columns from the dataset
df2.drop(['weather_code','is_holiday','is_weekend','season','month',
          'year','time_hour'],axis=1, inplace=True)
```

In [52]:

```
# update the column names dot to underscore
df2.columns = df2.columns.str.replace(".", "_")
df2.head()
```

Out[52]:

	cnt	t1	hum	wind_speed	wthr_2_0	wthr_3_0	wthr_4_0	wthr_7_0	wthr_10_0	wthr_26_0	...	time_14	time_15	time_16	time_17	time_18
0	182.0	3.0	93.0	6.0	0	1	0	0	0	0	...	0	0	0	0	0
1	138.0	3.0	93.0	5.0	0	0	0	0	0	0	...	0	0	0	0	0
2	134.0	2.5	96.5	0.0	0	0	0	0	0	0	...	0	0	0	0	0
3	72.0	2.0	100.0	0.0	0	0	0	0	0	0	...	0	0	0	0	0
4	47.0	2.0	93.0	6.5	0	0	0	0	0	0	...	0	0	0	0	0

5 rows × 51 columns

2-2-3. Checking Modeling Result - Second Modeling

In [53]:

```
outcome = 'cnt'
prediction = df2.drop('cnt', axis=1)

pred_sum = '+'.join(prediction.columns)
formula = outcome + '~' + pred_sum

model = ols(formula=formula, data=df2).fit()
model.summary()
```

Out[53]:

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.611
Model:	OLS	**Adj. R-squared:**	0.610
Method:	Least Squares	**F-statistic:**	579.6
Date:	Sat, 05 Aug 2023	**Prob (F-statistic):**	0.00
Time:	12:48:09	**Log-Likelihood:**	-1.3440e+05
No. Observations:	17414	**AIC:**	2.689e+05
Df Residuals:	17366	**BIC:**	2.693e+05
Df Model:	47		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	654.8945	39.113	16.744	0.000	578.230	731.559
t1	23.7198	1.540	15.398	0.000	20.700	26.739
hum	-8.0953	0.455	-17.799	0.000	-8.987	-7.204
wind_speed	-5.8193	0.652	-8.919	0.000	-7.098	-4.540
wthr_2_0	51.6245	11.762	4.389	0.000	28.570	74.679
wthr_3_0	19.7762	12.857	1.538	0.124	-5.425	44.977
wthr_4_0	-9.2175	16.693	-0.552	0.581	-41.937	23.502
wthr_7_0	-201.5944	15.705	-12.837	0.000	-232.377	-170.811
wthr_10_0	-640.9214	146.296	-4.381	0.000	-927.676	-354.167
wthr_26_0	31.9892	71.458	0.448	0.654	-108.076	172.054
hol_1_0	-174.0937	28.953	-6.013	0.000	-230.845	-117.342
wkd_1_0	-72.0074	9.234	-7.798	0.000	-90.106	-53.908
ssn_1_0	94.2218	12.623	7.464	0.000	69.480	118.964
ssn_2_0	95.2421	9.966	9.557	0.000	75.708	114.776
ssn_3_0	72.8325	15.958	4.564	0.000	41.553	104.112
mth_2	14.9677	20.805	0.719	0.472	-25.812	55.747
mth_3	96.2049	14.454	6.656	0.000	67.873	124.537
mth_4	147.0665	13.943	10.548	0.000	119.737	174.396
mth_5	149.3268	15.286	9.769	0.000	119.365	179.289
mth_6	44.9722	12.135	3.706	0.000	21.186	68.758
mth_7	46.1725	12.601	3.664	0.000	21.474	70.871
mth_8	3.0771	12.791	0.241	0.810	-21.995	28.149
mth_9	-0.4511	13.854	-0.033	0.974	-27.607	26.705
mth_10	83.7147	12.205	6.859	0.000	59.792	107.637
mth_11	11.9786	12.931	0.926	0.354	-13.368	37.325
mth_12	-21.5696	21.289	-1.013	0.311	-63.297	20.158
yr_2016	22.8480	8.335	2.741	0.006	6.511	39.185
yr_2017	-76.8131	66.725	-1.151	0.250	-207.601	53.975
time_1	-75.0200	28.639	-2.620	0.009	-131.155	-18.885
time_2	-127.3419	28.690	-4.439	0.000	-183.576	-71.107
time_3	-161.1089	28.701	-5.613	0.000	-217.365	-104.853
time_4	-170.1819	28.718	-5.926	0.000	-226.473	-113.891
time_5	-129.0862	28.736	-4.492	0.000	-185.412	-72.760
time_6	213.8858	28.725	7.446	0.000	157.583	270.189
time_7	1180.2182	28.698	41.125	0.000	1123.967	1236.469
time_8	912.7235	28.725	31.774	0.000	856.419	969.028
time_9	1288.9610	28.822	44.721	0.000	1232.467	1345.455
time_10	663.6255	29.075	22.824	0.000	606.635	720.616
time_11	710.2369	29.362	24.189	0.000	652.685	767.789
time_12	971.3485	29.573	32.846	0.000	913.382	1029.315
time_13	1004.2604	29.721	33.789	0.000	946.004	1062.517

time_14	948.5311	29.801	31.828	0.000	890.117	1006.945
time_15	1030.9192	29.733	34.673	0.000	972.640	1089.199
time_16	1344.1041	29.614	45.387	0.000	1286.057	1402.151
time_17	1566.2438	29.434	53.211	0.000	1508.549	1623.938
time_18	1553.3321	29.203	53.190	0.000	1496.091	1610.574
time_19	1232.6581	28.999	42.507	0.000	1175.818	1289.498
time_20	672.1861	28.825	23.319	0.000	615.686	728.686
time_21	388.4395	28.705	13.532	0.000	332.175	444.704
time_22	265.1952	28.655	9.255	0.000	209.029	321.362
time_23	130.4151	28.660	4.550	0.000	74.239	186.591

Omnibus:	2475.000	**Durbin-Watson:**	1.132
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	10211.706
Skew:	0.657	**Prob(JB):**	0.00
Kurtosis:	6.514	**Cond. No.**	1.78e+17

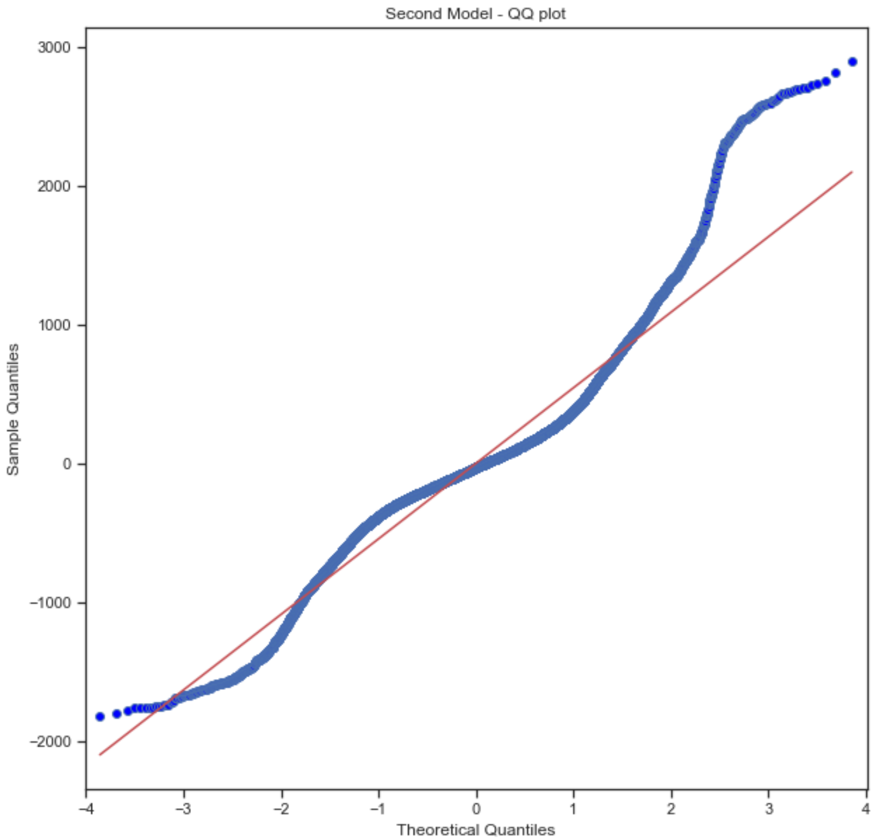
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.22e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Second Model Analysis

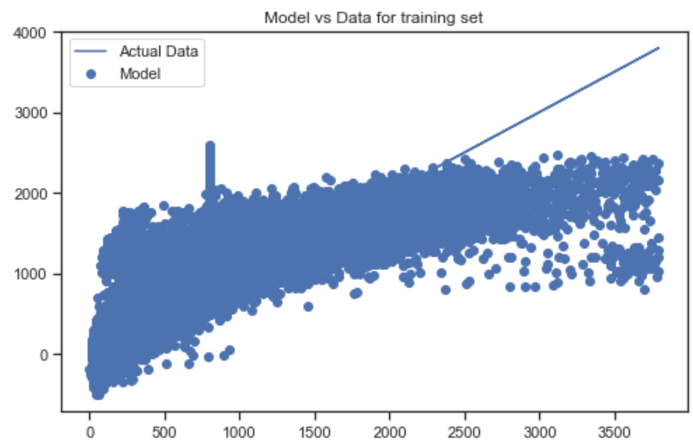
- The R-squared score has been improved from 0.312 to 0.611
- Skewness improved a lot close to 0, but Kurtosis is
- Kurtosis is more than 6, which means it is a leptokurtic

In [54]:

```
residuals = model.resid

fig, ax = plt.subplots(1,1)
fig.set_figheight(10)
fig.set_figwidth(10)

sm.ProbPlot(residuals).qqplot(line='s',ax=ax)
ax.title.set_text('Second Model - QQ plot');
```



In [55]:
```python
y = df2[['cnt']]
X = df2.drop(['cnt'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.3)

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
train_prediction = linreg.predict(X_train)
test_prediction = linreg.predict(X_test)

plt.figure(figsize=(8,5))
plt.scatter(y_train, train_prediction, label='Model')
plt.plot(y_train, y_train, label='Actual Data')
plt.title('Model vs Data for training set')
plt.legend();
```



2-3. Third Modeling

2-3-1. Log Transformation & Scaling - Normalising Data

In [56]:
```python
df3 = df2.copy()
```

In [57]:
```python
df3 = df3.reset_index()
```

In [58]:
```python
#conduct log transformation of the numeric values
a = df3['cnt']
b = df3['t1']
c = df3['hum']
d = df3['wind_speed']

logcnt = np.log(a, where = a>0)
logtemp = np.log(b, where = b>0)
loghum = np.log(c, where = c>0)
logwind = np.log(d, where = d>0)
```

In [59]:
```python
#conduct scaling of the numeric values
scaled_cnt = (logcnt-np.mean(logcnt))/np.sqrt(np.var(logcnt))
scaled_temp = (logtemp-np.mean(logtemp))/np.sqrt(np.var(logtemp))
scaled_hum = (loghum-np.mean(loghum))/np.sqrt(np.var(loghum))
scaled_wind = (logwind-np.mean(logwind))/np.sqrt(np.var(logwind))
```

In [60]:
```python
df3['cnt'] =scaled_cnt
df3['t1'] = scaled_temp
df3['hum'] = scaled_hum
df3['wind_speed'] = scaled_wind
```

In [61]:
```python
df3 = df3.drop('index', axis=1)
```

2-3-2. Checking Modeling Result - Third Modeling

In [62]:

```
outcome = 'cnt'
predictors = df3.drop('cnt', axis=1)
pred_sum = '+'.join(predictors.columns)
formula = outcome + '~' + pred_sum

model = ols(formula=formula, data=df3).fit()
model.summary()
```

Out[62]:

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.807
Model:	OLS	**Adj. R-squared:**	0.806
Method:	Least Squares	**F-statistic:**	1543.
Date:	Sat, 05 Aug 2023	**Prob (F-statistic):**	0.00
Time:	12:48:10	**Log-Likelihood:**	-10393.
No. Observations:	17414	**AIC:**	2.088e+04
Df Residuals:	17366	**BIC:**	2.126e+04
Df Model:	47		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	9.776e+10	2.83e+11	0.346	0.729	-4.56e+11	6.52e+11
t1	0.0620	0.006	11.186	0.000	0.051	0.073
hum	-0.0532	0.005	-10.827	0.000	-0.063	-0.044
wind_speed	0.0022	0.003	0.661	0.509	-0.004	0.009
wthr_2_0	0.0155	0.009	1.644	0.100	-0.003	0.034
wthr_3_0	-0.0065	0.010	-0.636	0.525	-0.027	0.014
wthr_4_0	-0.0413	0.013	-3.063	0.002	-0.068	-0.015
wthr_7_0	-0.3143	0.012	-25.386	0.000	-0.339	-0.290
wthr_10_0	-0.6295	0.118	-5.327	0.000	-0.861	-0.398
wthr_26_0	-0.4308	0.058	-7.473	0.000	-0.544	-0.318
hol_1_0	-0.1874	0.023	-8.013	0.000	-0.233	-0.142
wkd_1_0	-0.0115	0.007	-1.546	0.122	-0.026	0.003
ssn_1_0	-2.155e+11	6.23e+11	-0.346	0.729	-1.44e+12	1.01e+12
ssn_2_0	-2.133e+11	6.16e+11	-0.346	0.729	-1.42e+12	9.95e+11
ssn_3_0	-9.776e+10	2.83e+11	-0.346	0.729	-6.52e+11	4.56e+11
mth_2	0.0372	0.017	2.215	0.027	0.004	0.070
mth_3	-9.776e+10	2.83e+11	-0.346	0.729	-6.52e+11	4.56e+11
mth_4	-9.776e+10	2.83e+11	-0.346	0.729	-6.52e+11	4.56e+11
mth_5	-9.776e+10	2.83e+11	-0.346	0.729	-6.52e+11	4.56e+11
mth_6	1.178e+11	3.4e+11	0.346	0.729	-5.49e+11	7.85e+11
mth_7	1.178e+11	3.4e+11	0.346	0.729	-5.49e+11	7.85e+11
mth_8	1.178e+11	3.4e+11	0.346	0.729	-5.49e+11	7.85e+11
mth_9	1.155e+11	3.34e+11	0.346	0.729	-5.39e+11	7.7e+11
mth_10	1.155e+11	3.34e+11	0.346	0.729	-5.39e+11	7.7e+11
mth_11	1.155e+11	3.34e+11	0.346	0.729	-5.39e+11	7.7e+11
mth_12	0.0715	0.017	4.161	0.000	0.038	0.105
yr_2016	0.0123	0.007	1.837	0.066	-0.001	0.026
yr_2017	-0.2366	0.054	-4.383	0.000	-0.342	-0.131
time_1	-0.3611	0.023	-15.602	0.000	-0.406	-0.316
time_2	-0.6922	0.023	-29.854	0.000	-0.738	-0.647
time_3	-1.0041	0.023	-43.305	0.000	-1.050	-0.959
time_4	-1.0766	0.023	-46.388	0.000	-1.122	-1.031
time_5	-0.6533	0.023	-28.154	0.000	-0.699	-0.608
time_6	0.3292	0.023	14.188	0.000	0.284	0.375
time_7	1.1170	0.023	48.160	0.000	1.071	1.162
time_8	1.0078	0.023	43.410	0.000	0.962	1.053
time_9	1.4106	0.023	60.575	0.000	1.365	1.456
time_10	1.0939	0.023	46.564	0.000	1.048	1.140
time_11	1.1030	0.024	46.505	0.000	1.057	1.149
time_12	1.2775	0.024	53.498	0.000	1.231	1.324
time_13	1.2920	0.024	53.806	0.000	1.245	1.339

time_14	1.2501	0.024	51.921	0.000	1.203	1.297
time_15	1.2994	0.024	54.099	0.000	1.252	1.347
time_16	1.4762	0.024	61.710	0.000	1.429	1.523
time_17	1.4969	0.024	62.977	0.000	1.450	1.543
time_18	1.4917	0.024	63.270	0.000	1.446	1.538
time_19	1.3950	0.023	59.583	0.000	1.349	1.441
time_20	1.0488	0.023	45.044	0.000	1.003	1.094
time_21	0.7903	0.023	34.077	0.000	0.745	0.836
time_22	0.6313	0.023	27.267	0.000	0.586	0.677
time_23	0.3853	0.023	16.633	0.000	0.340	0.431

Omnibus:	763.243	**Durbin-Watson:**	0.656
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1630.791
Skew:	-0.297	**Prob(JB):**	0.00
Kurtosis:	4.376	**Cond. No.**	9.79e+15

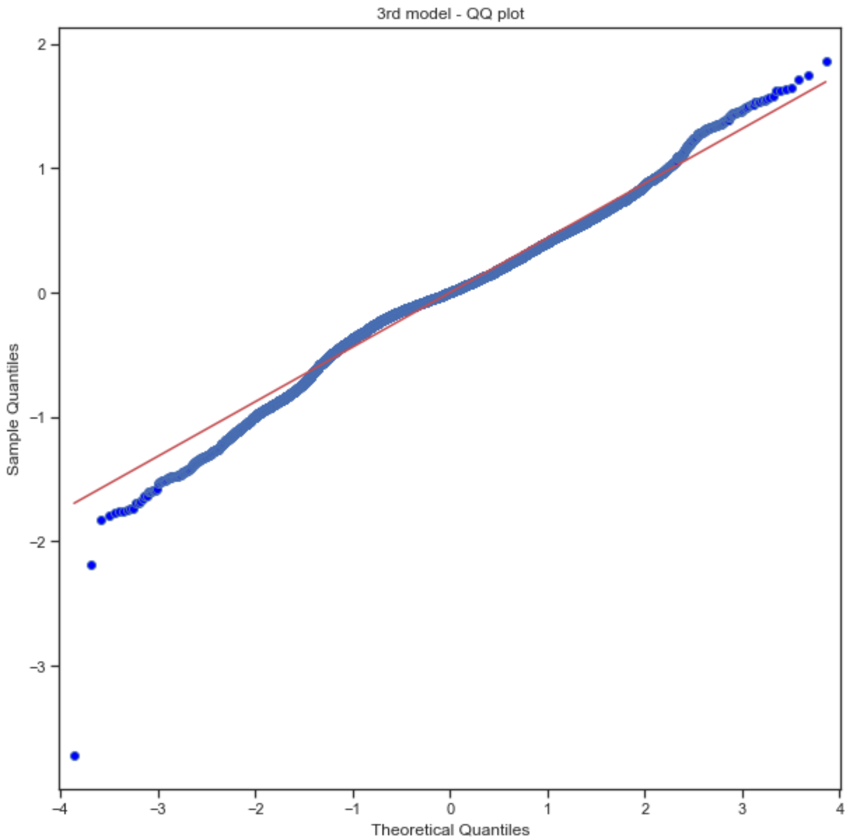
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.29e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Third Model Analysis

- The R-squared score has been improved to 0.807 from 0.611
- Skewness has been improved closer to 0 as -0.297
- Kurtosis also has been improved closer to 3
- Can see some of the variables are not statistically significant which is to be dropped for the final model

In [63]:

```
residuals = model.resid

fig, ax = plt.subplots(1,1)
fig.set_figheight(10)
fig.set_figwidth(10)

sm.ProbPlot(residuals).qqplot(line='s',ax=ax)
ax.title.set_text('3rd model - QQ plot');
```



In [64]:

```
y = df3[['cnt']]
X = df3.drop(['cnt'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.3)

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
train_prediction = linreg.predict(X_train)
test_prediction = linreg.predict(X_test)

plt.figure(figsize=(8,5))
plt.scatter(y_train, train_prediction, label='Model')
plt.plot(y_train, y_train, label='Actual Data')
plt.title('Model vs Data for training set')
plt.legend();
#Can observe some outliers
```

2-4. Final Modeling

2-4-1. Removing the variables that have p-value > 0.05

In [65]:

```
#final modelling
df4 = df3.drop(columns = ['wind_speed','wthr_2_0','wthr_3_0','wkd_1_0','ssn_1_0','ssn_2_0','ssn_3_0',
                          'mth_3','mth_4','mth_5','mth_6','mth_7','mth_8','mth_9','mth_10','mth_11',
                          'yr_2016','yr_2017'])
```

2-4-2. Checking Modeling Result - Final Modeling

In [66]:

```
outcome = 'cnt'
predictors = df4.drop('cnt', axis=1)
pred_sum = '+'.join(predictors.columns)
formula = outcome + '~' + pred_sum

model = ols(formula=formula, data=df4).fit()
model.summary()
```

Out[66]:

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.802
Model:	OLS	**Adj. R-squared:**	0.801
Method:	Least Squares	**F-statistic:**	2197.
Date:	Sat, 05 Aug 2023	**Prob (F-statistic):**	0.00
Time:	12:48:11	**Log-Likelihood:**	-10618.
No. Observations:	17414	**AIC:**	2.130e+04
Df Residuals:	17381	**BIC:**	2.156e+04
Df Model:	32		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.5957	0.017	-35.474	0.000	-0.629	-0.563
t1	0.1366	0.004	34.521	0.000	0.129	0.144
hum	-0.0572	0.004	-12.951	0.000	-0.066	-0.049
wthr_4_0	-0.0552	0.013	-4.344	0.000	-0.080	-0.030
wthr_7_0	-0.3370	0.011	-30.714	0.000	-0.359	-0.316
wthr_10_0	-0.6189	0.119	-5.185	0.000	-0.853	-0.385
wthr_26_0	-0.4454	0.058	-7.664	0.000	-0.559	-0.332
hol_1_0	-0.2082	0.023	-9.000	0.000	-0.254	-0.163
mth_2	-0.0418	0.014	-3.044	0.002	-0.069	-0.015
mth_12	-0.0750	0.013	-5.939	0.000	-0.100	-0.050
time_1	-0.3579	0.023	-15.278	0.000	-0.404	-0.312
time_2	-0.6846	0.023	-29.180	0.000	-0.731	-0.639
time_3	-0.9934	0.023	-42.339	0.000	-1.039	-0.947
time_4	-1.0636	0.023	-45.317	0.000	-1.110	-1.018
time_5	-0.6393	0.023	-27.229	0.000	-0.685	-0.593
time_6	0.3405	0.023	14.512	0.000	0.294	0.386
time_7	1.1240	0.023	47.942	0.000	1.078	1.170
time_8	1.0045	0.023	42.836	0.000	0.959	1.050
time_9	1.3971	0.023	59.584	0.000	1.351	1.443
time_10	1.0689	0.024	45.393	0.000	1.023	1.115
time_11	1.0695	0.024	45.203	0.000	1.023	1.116
time_12	1.2377	0.024	52.052	0.000	1.191	1.284
time_13	1.2481	0.024	52.249	0.000	1.201	1.295
time_14	1.2039	0.024	50.237	0.000	1.157	1.251
time_15	1.2542	0.024	52.387	0.000	1.207	1.301
time_16	1.4340	0.024	59.994	0.000	1.387	1.481
time_17	1.4586	0.024	61.272	0.000	1.412	1.505
time_18	1.4575	0.024	61.512	0.000	1.411	1.504
time_19	1.3669	0.024	57.919	0.000	1.321	1.413
time_20	1.0260	0.024	43.641	0.000	0.980	1.072
time_21	0.7744	0.023	33.019	0.000	0.728	0.820
time_22	0.6206	0.023	26.486	0.000	0.575	0.667
time_23	0.3803	0.023	16.221	0.000	0.334	0.426

Omnibus:	829.407	**Durbin-Watson:**	0.648
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1695.104
Skew:	-0.338	**Prob(JB):**	0.00
Kurtosis:	4.370	**Cond. No.**	41.5

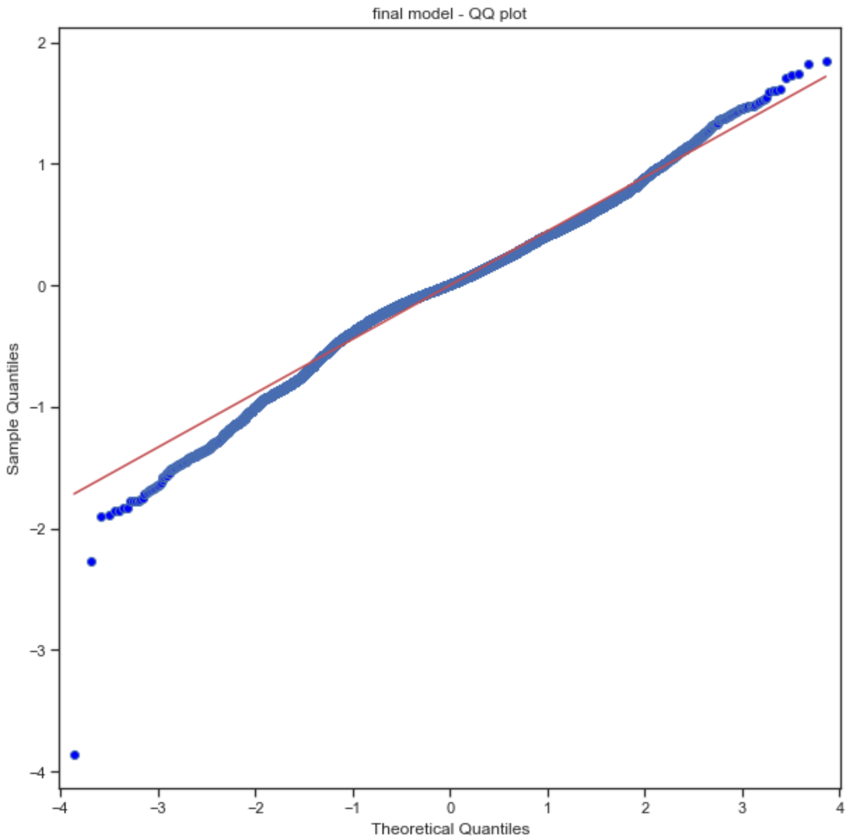
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Final Model Analysis

- The p-value of the intercept has been adjusted to 0
- Skewness and Kurtosis have been improved
- The R squared score got a good score as 0.802

In [67]:

```
residuals = model.resid

fig, ax = plt.subplots(1,1)
fig.set_figheight(10)
fig.set_figwidth(10)

sm.ProbPlot(residuals).qqplot(line='s',ax=ax)
ax.title.set_text('final model - QQ plot');
```



In [68]:

```python
y = df4[['cnt']]
X = df4.drop(['cnt'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.3)

linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
train_prediction = linreg.predict(X_train)
test_prediction = linreg.predict(X_test)

plt.figure(figsize=(8,5))
plt.scatter(y_train, train_prediction, label='Model')
plt.plot(y_train, y_train, label='Actual Data')
plt.title('Model vs Data for training set')
plt.legend();
```

3. Model Validation

3-1. Train/Test Split Validation

In [69]:

```python
train, test = train_test_split(df4)
```

In [70]:

```python
y = df4[['cnt']]
X = df4.drop(['cnt'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.4)
```

In [71]:

```python
print(len(X_train), len(X_test), len(y_train),len(y_test))
```

```
10448 6966 10448 6966
```

In [72]:

```python
linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

Out[72]:

```
LinearRegression()
```

In [73]:

```python
y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
```

In [74]:

```python
train_residuals = y_hat_train - y_train
test_residuals = y_hat_test - y_test
```

In [75]:

```python
test_mse = mean_squared_error(y_test, y_hat_test)
train_mse = mean_squared_error(y_train, y_hat_train)
```

In [76]:

```python
print('Train MSE:', train_mse)
print('Test MSE:', test_mse)
# The values are very close to each other
```

```
Train MSE: 0.19954325025505731
Test MSE: 0.1977221869537297
```

3-2. Cross Validation

In [77]:
```
cv_mse = -cross_val_score(linreg, X_train, y_train, scoring='neg_mean_squared_error', cv=10)
print('K-fold cross validation MSE:', round(cv_mse.mean(),6))
#very close to the scores of train/test validation
```

```
K-fold cross validation MSE: 0.200782
```

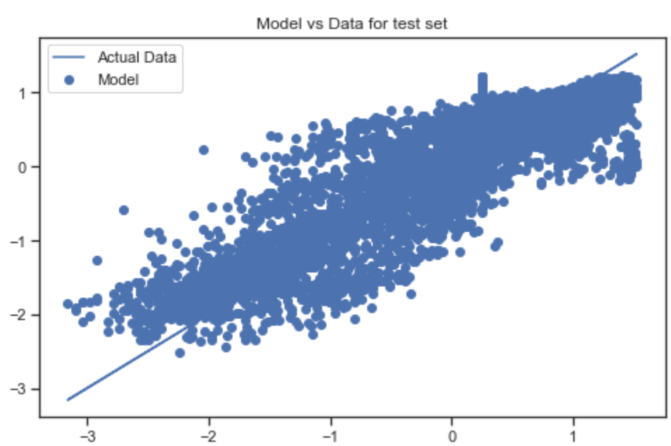
In [78]:
```
cv_r2 = cross_val_score(linreg, X_train, y_train, scoring='r2', cv=10)
print('K-fold cross validation R-2:', round(cv_r2.mean(),6))
#also very close to the scores of train/test validation
```

```
K-fold cross validation R-2: 0.795582
```

In [79]:
```
train_prediction = linreg.predict(X_train)
test_prediction = linreg.predict(X_test)
```

In [80]:
```
plt.figure(figsize=(8,5))
plt.scatter(y_train, train_prediction, label='Model')
plt.plot(y_train, y_train, label='Actual Data')
plt.title('Model vs Data for training set')
plt.legend();
```

In [81]:
```
plt.figure(figsize=(8,5))
plt.scatter(y_test, test_prediction, label='Model')
plt.plot(y_test, y_test, label='Actual Data')
plt.title('Model vs Data for test set')
plt.legend();
```



3-3. Checking Coefficient Scores

In [82]:
```
coeff = model.params
ranked_features = coeff.sort_values(ascending=False)
ranked_features
```

Out[82]:
```
time_17       1.458600
time_18       1.457478
time_16       1.433964
time_9        1.397131
time_19       1.366941
time_15       1.254191
time_13       1.248100
time_12       1.237737
time_14       1.203868
time_7        1.123958
time_11       1.069475
time_10       1.068877
time_20       1.025992
time_8        1.004514
time_21       0.774370
time_22       0.620581
time_23       0.380265
time_6        0.340471
t1            0.136606
mth_2        -0.041818
wthr_4_0     -0.055169
hum          -0.057241
mth_12       -0.075043
hol_1_0      -0.208220
wthr_7_0     -0.337036
time_1       -0.357886
wthr_26_0    -0.445433
Intercept    -0.595705
wthr_10_0    -0.618901
time_5       -0.639302
time_2       -0.684590
time_3       -0.993429
time_4       -1.063613
dtype: float64
```

4. Result

1. The most effective features based on multiple linear regression analysis:

- Time: the rush hour & during the days generate more bike hire counts than the rest of the times
- Weather: bad weather (raining and snowing) affects to reduce the number of bike hire
- Temp: higher temperatures makes Londoners hire bikes

2. Additionally, what we can add from the general analysis observation:

- Higher demand on Weekdays compare to weekends and public holidays - bike hire is loved by Londoners more than tourists
- Summer is the most popular season to hire bikes

5. Further action

- Obtain the location data analysis: To supply the right amount of bikes to hire in the right places, by investigating the bike trip flow based on the location data, we will be able to predict not only the demand but also effective bike-relocation to increase the bike hire
- Adopt a new type of vehicle and demand: E-bikes and e-scooter were released in late 2020 in London, we should look into the recent data to see the trends to predict more accurate customer demands