Bike Sharing Demand: Linear Regression Model

Problem Statement:

A bike-sharing system is a service in which bikes are made available for shared use to individuals on a short term basis for a price or free. Many bike share systems allow people to borrow a bike from a "dock" which is usually computer-controlled wherein the user enters the payment information, and the system unlocks it. This bike can then be returned to another dock belonging to the same system.

A US bike-sharing provider **BoomBikes** has recently suffered considerable dips in their revenues due to the ongoing Corona pandemic.

They want to understand the factors on which the demand for these shared bikes depends. Specifically, they want to understand the factors affecting the demand for these shared bikes in the American market. The company wants to know:

- Which variables are significant in predicting the demand for shared bikes.
- How well those variables describe the bike demands

Business Goal

We need build a model to predict the demand for shared bikes with the available independent variables in the given dataset. It will be used by the management to understand how exactly the demands vary with different features. They can accordingly manipulate the business strategy to meet the demand levels and meet the customer's expectations. Further, the model will be a good way for management to understand the demand dynamics of a new market.

Steps followed in this exercise

Broadly, we will follow the following steps to build the model:

1. Reading, understanding and visualizing the data
2. Preparing the data for model (train-test split, rescaling etc.)
3. Training the model
4. Residual Analysis
5. Predictions and evaluations on the test set

Reading and Understanding the data

In [412]:

```
# Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [413]:

```python
# Read dataset
data = pd.read_csv("day.csv")
pd.concat([data.head(3), data.tail(3)])
```

Out[413]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	w
0	1	01-01-2018	1	0	1	0	6	0	2	14.110847	18.18125	80.5833	
1	2	02-01-2018	1	0	1	0	0	0	2	14.902598	17.68695	69.6087	
2	3	03-01-2018	1	0	1	0	1	1	1	8.050924	9.47025	43.7273	
727	728	29-12-2019	1	1	12	0	6	0	2	10.386653	12.12000	75.2917	
728	729	30-12-2019	1	1	12	0	0	0	1	10.489153	11.58500	48.3333	
729	730	31-12-2019	1	1	12	0	1	1	2	8.849153	11.17435	57.7500	

In [414]:

```python
# Check dimension
print("Shape of data:", data.shape)
```

```
Shape of data: (730, 16)
```

In [415]:

```python
# Get basic information about the columns and their data types
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   instant     730 non-null    int64  
 1   dteday      730 non-null    object 
 2   season      730 non-null    int64  
 3   yr          730 non-null    int64  
 4   mnth        730 non-null    int64  
 5   holiday     730 non-null    int64  
 6   weekday     730 non-null    int64  
 7   workingday  730 non-null    int64  
 8   weathersit  730 non-null    int64  
 9   temp        730 non-null    float64
 10  atemp       730 non-null    float64
 11  hum         730 non-null    float64
 12  windspeed   730 non-null    float64
 13  casual      730 non-null    int64  
 14  registered  730 non-null    int64  
 15  cnt         730 non-null    int64  
dtypes: float64(4), int64(11), object(1)
memory usage: 91.4+ KB
```

In [416]:

```
# Get descriptive statistics of data
print("Description statisctics of data", data.describe())
```

```
Description statisctics of data          instant      season          yr        mnt
h     holiday     weekday  \
count  730.000000  730.000000  730.000000  730.000000  730.000000  730.000000
mean   365.500000    2.498630    0.500000    6.526027    0.028767    2.997260
std    210.877136    1.110184    0.500343    3.450215    0.167266    2.006161
min      1.000000    1.000000    0.000000    1.000000    0.000000    0.000000
25%    183.250000    2.000000    0.000000    4.000000    0.000000    1.000000
50%    365.500000    3.000000    0.500000    7.000000    0.000000    3.000000
75%    547.750000    3.000000    1.000000   10.000000    0.000000    5.000000
max    730.000000    4.000000    1.000000   12.000000    1.000000    6.000000

       workingday  weathersit        temp       atemp         hum   windspeed  \
count  730.000000  730.000000  730.000000  730.000000  730.000000  730.000000
mean     0.683562    1.394521   20.319259   23.726322   62.765175   12.763620
std      0.465405    0.544807    7.506729    8.150308   14.237589    5.195841
min      0.000000    1.000000    2.424346    3.953480    0.000000    1.500244
25%      0.000000    1.000000   13.811885   16.889713   52.000000    9.041650
50%      1.000000    1.000000   20.465826   24.368225   62.625000   12.125325
75%      1.000000    2.000000   26.880615   30.445775   72.989575   15.625589
max      1.000000    3.000000   35.328347   42.044800   97.250000   34.000021

            casual   registered          cnt
count   730.000000   730.000000   730.000000
mean    849.249315  3658.757534  4508.006849
std     686.479875  1559.758728  1936.011647
min       2.000000    20.000000    22.000000
25%     316.250000  2502.250000  3169.750000
50%     717.000000  3664.500000  4548.500000
75%    1096.500000  4783.250000  5966.000000
max    3410.000000  6946.000000  8714.000000
```

Data Quality Check

Null value check

In [418]:

```
# Check if there are null values in dataset
data.isnull().any().any()
```

Out[418]:

```
False
```

There is no null value in dataset

Fixing weekday and workingday columns

Seems like weekday and workingday columns are shifted 2 steps down. This can be verified by dteday column. For example 1st Jan, 2018 is Monday and it should be working day.

In [419]:

```python
steps = 2
while steps > 0:
    val_1 = data.weekday.iloc[0]
    val_2 = data.workingday.iloc[0]
    data['weekday'] = data['weekday'].shift(-1).fillna(0).astype(int)
    data['workingday'] = data['workingday'].shift(-1).fillna(0).astype(int)
    data.at[data.index[-1], 'weekday'] = val_1
    data.at[data.index[-1], 'workingday'] = val_2
    steps = steps - 1
pd.concat([data.head(3), data.tail(3)])
```

Out[419]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	w
0	1	01-01-2018	1	0	1	0	1	1	2	14.110847	18.18125	80.5833	
1	2	02-01-2018	1	0	1	0	2	1	2	14.902598	17.68695	69.6087	
2	3	03-01-2018	1	0	1	0	3	1	1	8.050924	9.47025	43.7273	
727	728	29-12-2019	1	1	12	0	1	1	2	10.386653	12.12000	75.2917	
728	729	30-12-2019	1	1	12	0	6	0	1	10.489153	11.58500	48.3333	
729	730	31-12-2019	1	1	12	0	0	0	2	8.849153	11.17435	57.7500	

Data cleaning

In [420]:

```python
# Drop column 'instant' since it is an identity/index column
data = data.drop('instant', axis= 1)

# Converting dteday to Pandas datetime format
data['dteday'] = pd.to_datetime(data['dteday'], dayfirst= True)

# Droping column 'cnt' gives summation of casual and registered values. So we can drop column
data = data.drop(['casual', 'registered'], axis= 1)

# Derive a new column day which represent days passed since beginning of data collection
from datetime import date
start_date = date(2017, 12, 31)
data['day'] = (data.dteday.dt.date - start_date).dt.days

# Mapping season to actual values
data['season'] = data['season'].map({1: 'spring', 2: 'summer', 3: 'fall', 4: 'winter'})

pd.concat([data.head(3), data.tail(3)])
```

Out[420]:

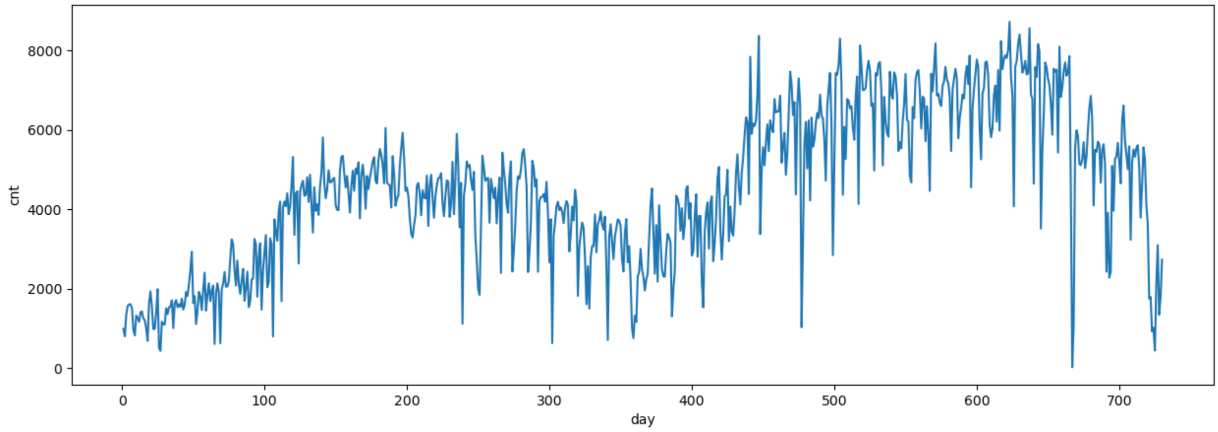
	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed
0	2018-01-01	spring	0	1	0	1	1	2	14.110847	18.18125	80.5833	10.74988
1	2018-01-02	spring	0	1	0	2	1	2	14.902598	17.68695	69.6087	16.65211
2	2018-01-03	spring	0	1	0	3	1	1	8.050924	9.47025	43.7273	16.63670
727	2019-12-29	spring	1	12	0	1	1	2	10.386653	12.12000	75.2917	8.33366
728	2019-12-30	spring	1	12	0	6	0	1	10.489153	11.58500	48.3333	23.50051
729	2019-12-31	spring	1	12	0	0	0	2	8.849153	11.17435	57.7500	10.37468

Visualizing data

Visualize count of bike rentals over time

In [421]:

```
fig = plt.figure(figsize=(15, 5))
sns.lineplot(data= data, x= 'day', y= 'cnt')
plt.show()
```

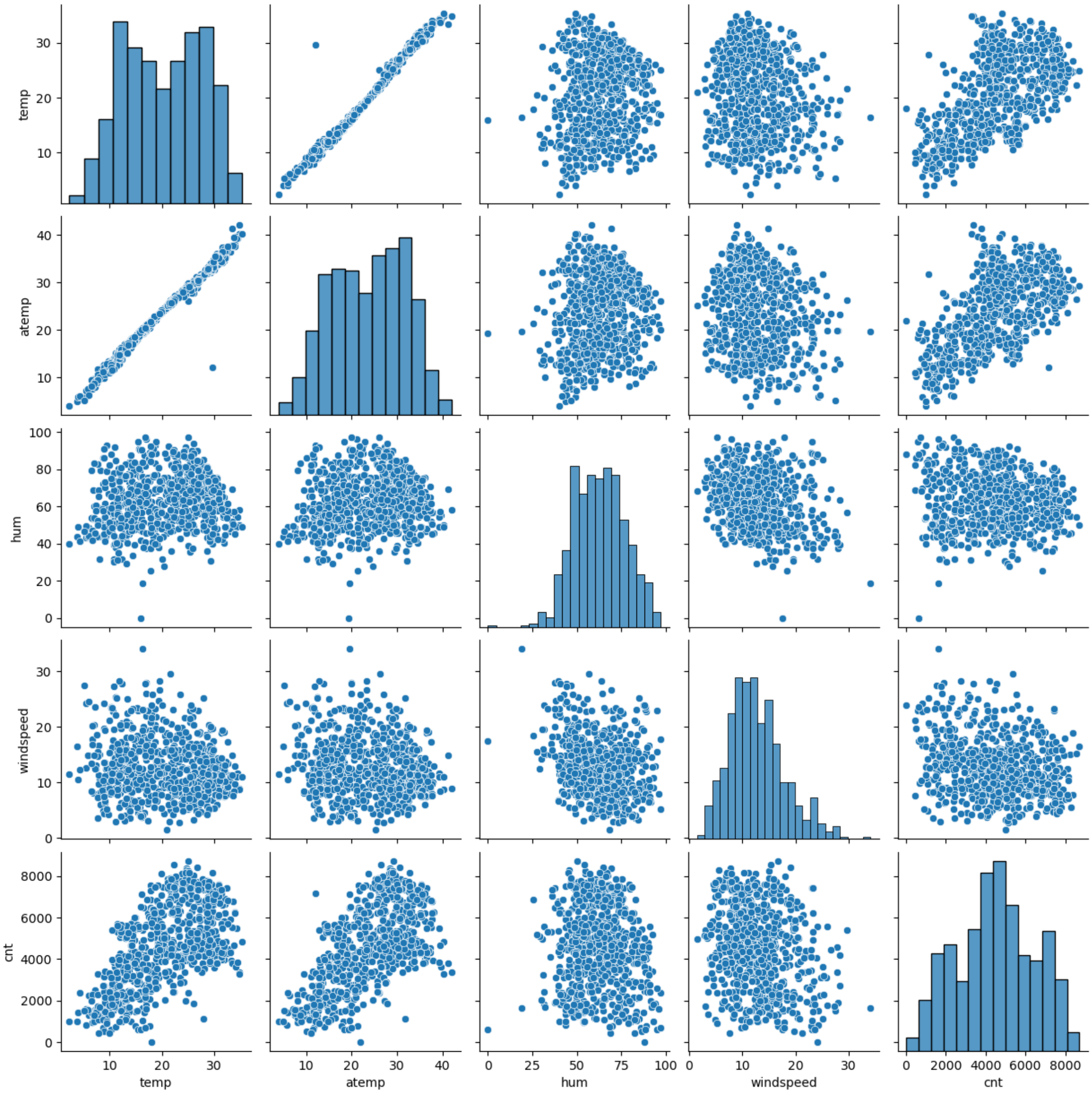


There is a considerable dip in bike rental demand towars the tail of the graph. Which makes sense because there was a COVID-19 impact on the business

Visualize numeric variables

In [422]:

```
numeric_vars = ['temp', 'atemp', 'hum', 'windspeed', 'cnt']
sns.pairplot(data[numeric_vars])
plt.show()
```

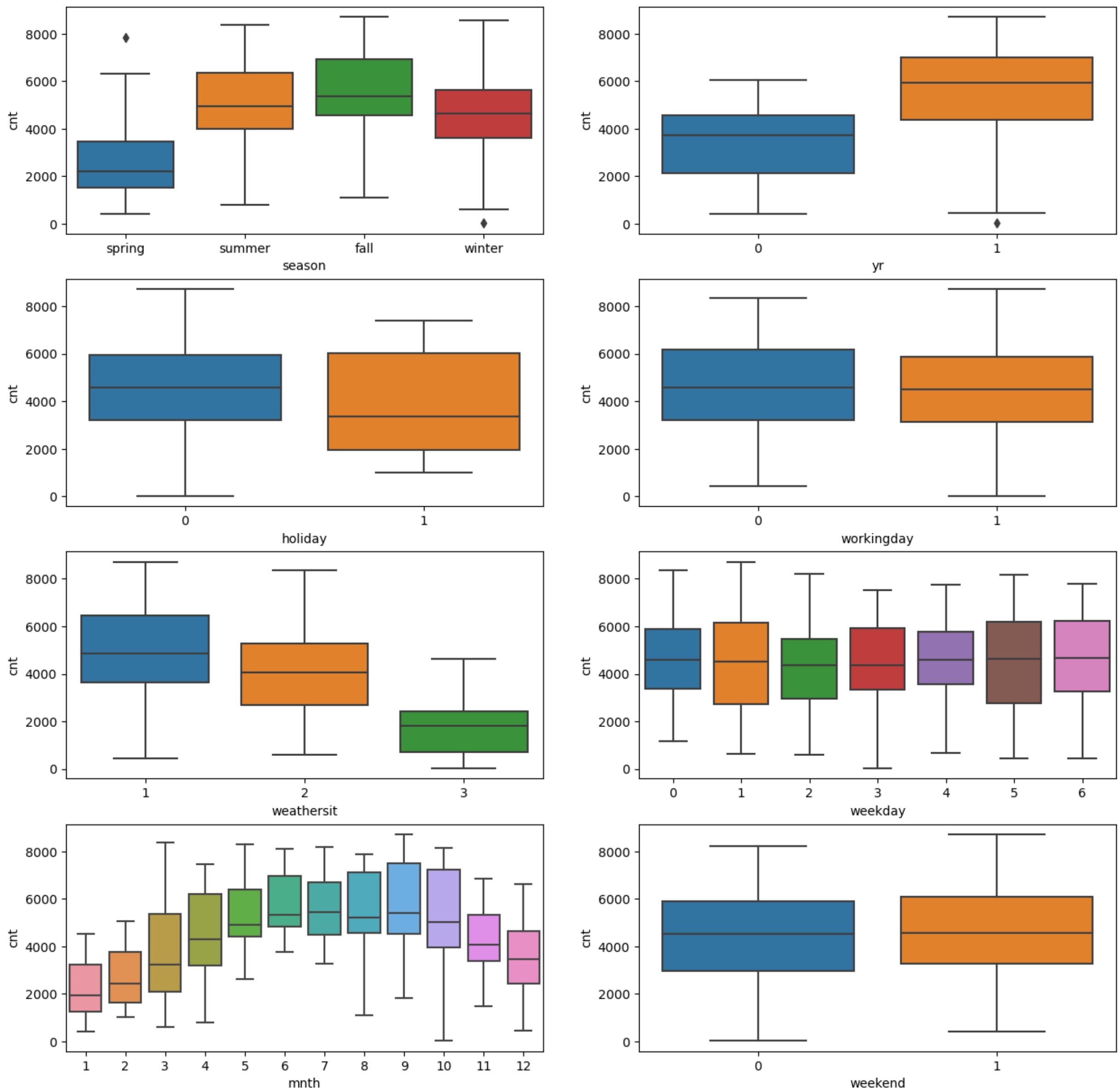


Linear relationships between temp and cnt, atemp and cnt, are clearly visible in the pairplot

Visualize categorical variables

In [423]:

```
# Introducing a derived variable 'weekend'
data['weekend'] = data['dteday'].dt.day_name().isin(['Saturday', 'Sunday']).astype(int)

categorical_vars = ['season', 'yr', 'holiday', 'workingday', 'weathersit', 'weekday', 'mnth',

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

for index, var in enumerate(categorical_vars):
    plt.subplot(4, 2, index + 1)
    sns.boxplot(x= var, y= 'cnt', data= data)
plt.show()
```



A few observation from above boxplots:

- There are more demand of bikes during Summer and Fall seasons
- Overall there were more demands in 2019 as compered to 2018 which means company's YOY growth was good during years 2018 to 2019
- Weather situation also affects the business. Clearer the weather, better for the business
- There a pattern of increasing demands from the Month of March till September

Preparing the data for model

Get dummy variables for non-binary categorical variables

In [424]:

```
# Dummy varaibles for season dropping redundant variable
seasons = pd.get_dummies(data['season'], drop_first= True)
seasons.head()
```

Out[424]:

	spring	summer	winter
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0

In [425]:

```
# Mapping mnth to actual values
import calendar
d = dict(enumerate(calendar.month_abbr))
data['mnth'] = data['mnth'].map(d)

# Dummy varaibles for mnth
months = pd.get_dummies(data['mnth'], drop_first= True)
months.head()
```

Out[425]:

	Aug	Dec	Feb	Jan	Jul	Jun	Mar	May	Nov	Oct	Sep
0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0

In [426]:

```
# Mapping weekday to actual values
data['weekday'] = data['dteday'].dt.day_name()

# Dummy varaibles for mnth
weekdays = pd.get_dummies(data['weekday'], drop_first= True)
weekdays.head()
```

Out[426]:

	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
0	1	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	0	0	1
3	0	0	0	1	0	0
4	0	0	0	0	0	0

In [427]:

```
# Encoding weathersit. Dataset contains only three values for weathersit.
data['weathersit'] = data['weathersit'].map({1: 'Clear', 2: 'Mist/Cloud', 3: 'Light Snow'})

# Dummy varaibles for weathersit
weathersits = pd.get_dummies(data['weathersit'], drop_first= True)
weathersits.head()
```

Out[427]:

	Light Snow	Mist/Cloud
0	0	1
1	0	1
2	0	0
3	0	0
4	0	0

In [428]:

```
# Merge dummy variables and drop originals
data = pd.concat([data, seasons, months, weathersits], axis=1)
data.drop(['season', 'mnth', 'weekday', 'weathersit'], inplace= True, axis= 1)
```

In [429]:

```
# Drop day as it was ony useful to visualize demand trend of the time
data.drop('day', inplace= True, axis= 1)

# Drop column 'dteday' as we already have all components of date as a separate columns in dat
data = data.drop('dteday', axis= 1)

data.head()
```

Out[429]:

	yr	holiday	workingday	temp	atemp	hum	windspeed	cnt	weekend	spring	...	Jan	Jul	Jun	M
0	0	0	1	14.110847	18.18125	80.5833	10.749882	985	0	1	...	1	0	0	
1	0	0	1	14.902598	17.68695	69.6087	16.652113	801	0	1	...	1	0	0	
2	0	0	1	8.050924	9.47025	43.7273	16.636703	1349	0	1	...	1	0	0	
3	0	0	1	8.200000	10.60610	59.0435	10.739832	1562	0	1	...	1	0	0	
4	0	0	1	9.305237	11.46350	43.6957	12.522300	1600	0	1	...	1	0	0	

5 rows × 25 columns

Spit data into Train-Test

In [430]:

```
np.random.seed(0)
df_train, df_test = train_test_split(data, train_size= 0.7, random_state= 100)
print("Training data shape:", df_train.shape)
print("Test data shape:", df_test.shape)
```

```
Training data shape: (510, 25)
Test data shape: (220, 25)
```

Rescale features

In [431]:

```
# normalisation: (x - xmin) / (xmax - xmin)
scaler = MinMaxScaler()
# create a list of variables to be rescaled
scale_vars = ['temp', 'atemp', 'hum', 'windspeed', 'cnt']
# Fit on data
df_train[scale_vars] = scaler.fit_transform(df_train[scale_vars])
df_train.head()
```

Out[431]:

	yr	holiday	workingday	temp	atemp	hum	windspeed	cnt	weekend	spring	...	Jan	Jul	J
576	1	0	1	0.815169	0.766351	0.725633	0.264686	0.827658	0	0	...	0	1	
426	1	0	1	0.442393	0.438975	0.640189	0.255342	0.465255	1	1	...	0	0	
728	1	0	0	0.245101	0.200348	0.498067	0.663106	0.204096	0	1	...	0	0	
482	1	0	1	0.395666	0.391735	0.504508	0.188475	0.482973	1	0	...	0	0	
111	0	0	0	0.345824	0.318819	0.751824	0.380981	0.191095	1	0	...	0	0	

5 rows × 25 columns

In [432]:

```
df_train[scale_vars].describe()
```

Out[432]:

	temp	atemp	hum	windspeed	cnt
count	510.000000	510.000000	510.000000	510.000000	510.000000
mean	0.537440	0.513156	0.650480	0.320883	0.513499
std	0.225858	0.212410	0.145846	0.169803	0.224421
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.339853	0.332086	0.538643	0.199179	0.356420
50%	0.542596	0.529718	0.653714	0.296763	0.518638
75%	0.735215	0.688457	0.754830	0.414447	0.684710
max	1.000000	1.000000	1.000000	1.000000	1.000000

Train the model

In [433]:

```
# Create X and y
y_train = df_train.pop('cnt')
X_train = df_train
```

Recursive feature elimination

In [434]:

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

# Running RFE specifying 12 features to choose
lm = LinearRegression()
lm.fit(X_train, y_train)

rfe = RFE(lm, n_features_to_select = 12)
rfe = rfe.fit(X_train, y_train)

# Inspect RFE result
list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

Out[434]:

```
[('yr', True, 1),
 ('holiday', True, 1),
 ('workingday', False, 13),
 ('temp', True, 1),
 ('atemp', False, 6),
 ('hum', True, 1),
 ('windspeed', True, 1),
 ('weekend', False, 9),
 ('spring', True, 1),
 ('summer', True, 1),
 ('winter', True, 1),
 ('Aug', False, 8),
 ('Dec', False, 4),
 ('Feb', False, 5),
 ('Jan', False, 2),
 ('Jul', True, 1),
 ('Jun', False, 10),
 ('Mar', False, 12),
 ('May', False, 7),
 ('Nov', False, 3),
 ('Oct', False, 11),
 ('Sep', True, 1),
 ('Light Snow', True, 1),
 ('Mist/Cloud', True, 1)]
```

In [435]:

```
col = X_train.columns[rfe.support_]
print("Out of", X_train.columns.size, "features,",col.size, "selected by RFE are:", list(col)
```

```
Out of 24 features, 12 selected by RFE are: ['yr', 'holiday', 'temp', 'hum', 'windsp
eed', 'spring', 'summer', 'winter', 'Jul', 'Sep', 'Light Snow', 'Mist/Cloud']
```

Model 1

With 12 variables selected after RFE, building the 1st model using Stats Model library

In [436]:

```
X_train_rfe = X_train[col]
# Adding a constant
X_train_rfe_sm = sm.add_constant(X_train_rfe)
# Fit the model
lm = sm.OLS(y_train, X_train_rfe_sm).fit()
```

In [437]:

```
1 lm.summary()
```

Out[437]:

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.842
Model:	OLS	**Adj. R-squared:**	0.838
Method:	Least Squares	**F-statistic:**	220.6
Date:	Tue, 13 Jun 2023	**Prob (F-statistic):**	2.95e-190
Time:	23:21:22	**Log-Likelihood:**	509.29
No. Observations:	510	**AIC:**	-992.6
Df Residuals:	497	**BIC:**	-937.5
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.2848	0.034	8.258	0.000	0.217	0.353
yr	0.2294	0.008	28.208	0.000	0.213	0.245
holiday	-0.0969	0.026	-3.787	0.000	-0.147	-0.047
temp	0.5299	0.034	15.728	0.000	0.464	0.596
hum	-0.1726	0.038	-4.569	0.000	-0.247	-0.098
windspeed	-0.1822	0.026	-7.074	0.000	-0.233	-0.132
spring	-0.0564	0.021	-2.700	0.007	-0.097	-0.015
summer	0.0531	0.015	3.536	0.000	0.024	0.083
winter	0.0976	0.017	5.643	0.000	0.064	0.132
Jul	-0.0572	0.018	-3.123	0.002	-0.093	-0.021
Sep	0.0833	0.017	4.973	0.000	0.050	0.116
Light Snow	-0.2369	0.026	-8.983	0.000	-0.289	-0.185
Mist/Cloud	-0.0527	0.010	-5.017	0.000	-0.073	-0.032

Omnibus:	57.486	**Durbin-Watson:**	2.051
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	130.221
Skew:	-0.612	**Prob(JB):**	5.28e-29
Kurtosis:	5.151	**Cond. No.**	19.4

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

In [438]:

```python
# Check VIFs
def get_vifs(X):
    vif = pd.DataFrame()
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by= 'VIF', ascending= False)
    return vif

X = X_train_rfe
vif = get_vifs(X)
```

Model 2

In [439]:

```python
# Drop hum which has a high VIF

X = X.drop('hum', axis= 1)
X_train_sm = sm.add_constant(X)

# create and fit the model
lm = sm.OLS(y_train, X_train_sm).fit()

# check params
lm.summary()
```

Out[439]:

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.835
Model:	OLS	**Adj. R-squared:**	0.832
Method:	Least Squares	**F-statistic:**	229.6
Date:	Tue, 13 Jun 2023	**Prob (F-statistic):**	5.06e-187
Time:	23:21:25	**Log-Likelihood:**	498.80
No. Observations:	510	**AIC:**	-973.6
Df Residuals:	498	**BIC:**	-922.8
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1994	0.030	6.746	0.000	0.141	0.258
yr	0.2336	0.008	28.352	0.000	0.217	0.250
holiday	-0.0975	0.026	-3.736	0.000	-0.149	-0.046
temp	0.4910	0.033	14.770	0.000	0.426	0.556
windspeed	-0.1479	0.025	-5.887	0.000	-0.197	-0.099
spring	-0.0672	0.021	-3.175	0.002	-0.109	-0.026
summer	0.0465	0.015	3.051	0.002	0.017	0.076
winter	0.0817	0.017	4.730	0.000	0.048	0.116
Jul	-0.0521	0.019	-2.790	0.005	-0.089	-0.015
Sep	0.0768	0.017	4.517	0.000	0.043	0.110
Light Snow	-0.2842	0.025	-11.487	0.000	-0.333	-0.236
Mist/Cloud	-0.0802	0.009	-9.146	0.000	-0.097	-0.063

Omnibus:	59.182	**Durbin-Watson:**	2.051
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	134.016
Skew:	-0.629	**Prob(JB):**	7.92e-30
Kurtosis:	5.173	**Cond. No.**	17.3

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

In [440]:

```
# Check VIFs
vif = get_vifs(X)
vif
```

Out[440]:

	Features	VIF
2	temp	5.09
3	windspeed	4.60
5	summer	2.23
4	spring	2.08
0	yr	2.07
6	winter	1.78
7	Jul	1.58
10	Mist/Cloud	1.55
8	Sep	1.34
9	Light Snow	1.08
1	holiday	1.04

Model 3

In [441]:

```python
# All variables a statistically significant and temp has a high VIF.
# We will still keep temp since we know that temperature is a strong predictor of bike demand

# Dropping Jul having slightly high p-value
X = X.drop('Jul', axis= 1)
X_train_sm = sm.add_constant(X)

# create and fit the model
lm = sm.OLS(y_train, X_train_sm).fit()

# check params
lm.summary()
```

Out[441]:

OLS Regression Results

Dep. Variable:	cnt	**R-squared:**	0.833
Model:	OLS	**Adj. R-squared:**	0.829
Method:	Least Squares	**F-statistic:**	248.4
Date:	Tue, 13 Jun 2023	**Prob (F-statistic):**	1.47e-186
Time:	23:21:27	**Log-Likelihood:**	494.84
No. Observations:	510	**AIC:**	-967.7
Df Residuals:	499	**BIC:**	-921.1
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1909	0.030	6.447	0.000	0.133	0.249
yr	0.2341	0.008	28.237	0.000	0.218	0.250
holiday	-0.0963	0.026	-3.668	0.000	-0.148	-0.045
temp	0.4777	0.033	14.423	0.000	0.413	0.543
windspeed	-0.1481	0.025	-5.854	0.000	-0.198	-0.098
spring	-0.0554	0.021	-2.654	0.008	-0.096	-0.014
summer	0.0621	0.014	4.350	0.000	0.034	0.090
winter	0.0945	0.017	5.630	0.000	0.062	0.127
Sep	0.0910	0.016	5.566	0.000	0.059	0.123
Light Snow	-0.2850	0.025	-11.444	0.000	-0.334	-0.236
Mist/Cloud	-0.0787	0.009	-8.938	0.000	-0.096	-0.061

Omnibus:	63.413	**Durbin-Watson:**	2.085
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	142.384
Skew:	-0.674	**Prob(JB):**	1.21e-31
Kurtosis:	5.210	**Cond. No.**	17.2

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

In [442]:

```
# Check VIFs
vif = get_vifs(X)
vif
```

Out[442]:

	Features	VIF
3	windspeed	4.60
2	temp	3.84
0	yr	2.07
4	spring	1.99
5	summer	1.90
6	winter	1.63
9	Mist/Cloud	1.55
7	Sep	1.23
8	Light Snow	1.08
1	holiday	1.04

Final Model 3 has a very good Adjusted R-Squared of 0.83 and as per the VIF values, we don't have any multicollinearity in variables.

Residual Analysis

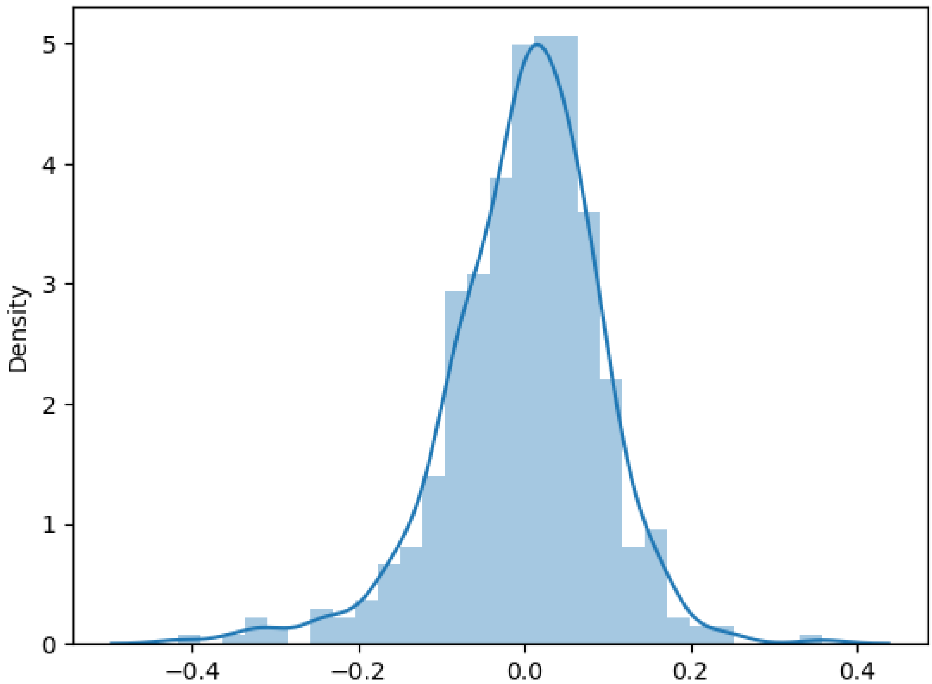
In [443]:

```
y_train_pred = lm.predict(X_train_sm)
res = y_train - y_train_pred
sns.distplot(res)
```

```
/var/folders/y8/s7bl9twj7gsc19zqg5ltypt00000gp/T/ipykernel_57614/3590158396.py:3: Us
erWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.githu
b.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(res)
```

Out[443]:

```
&lt;Axes: ylabel='Density'&gt;
```



Error terms are normally distributed with mean zero.

Making predictions

Scaling features on test data

In [444]:

```
# transform on data
df_test[scale_vars] = scaler.transform(df_test[scale_vars])
```

Creating X and y

In [445]:

```
y_test = df_test.pop('cnt')
X_test = df_test
```

Removing variables from Test data which were dropped in Training set

In [446]:

```
X_test_new = X_test[X.columns]
```

Prediction

In [447]:

```
# Add constant
X_test_new = sm.add_constant(X_test_new)

# Make prediction
y_test_pred = lm.predict(X_test_new)
```

Model evaluation

In [448]:

```
from sklearn.metrics import mean_squared_error, r2_score

mean_squared_error = mean_squared_error(y_test, y_test_pred)
r_squared = r2_score(y_test, y_test_pred)

print('Mean Squared Error :',mean_squared_error)
print('R Squared :', r_squared)
```

```
Mean Squared Error : 0.009380224523815579
R Squared : 0.8038195990728844
```

Calculating Adj. R-Squared

In [449]:

```
# number of rows in X_test_new
n = X_test_new.shape[0]

# number of columns in X_test_new
p = X_test_new.shape[1]

adjusted_r2 = 1 - (1 - r_squared) * (n - 1) / (n - p - 1)
adjusted_r2
```

Out[449]:

```
0.7934446740238542
```

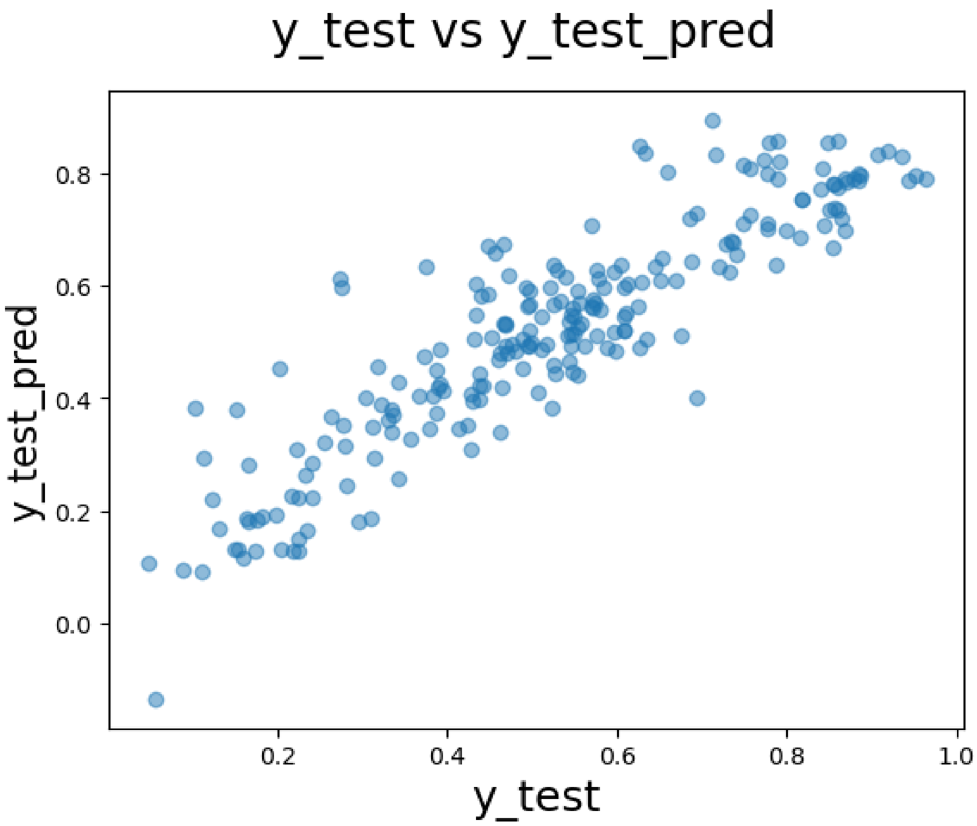
In [450]:

```
This value is very close to Adj.R-Squared of our final Model 3 which is 0.829
```

```
  Cell In[450], line 1
    This value is very close to Adj.R-Squared of our final Model 3 which is 0.829
         ^
SyntaxError: invalid syntax
```

In [451]:

```
# Plotting y_test and y_test_pred

fig = plt.figure()
plt.scatter(y_test, y_test_pred, alpha=.5)
fig.suptitle('y_test vs y_test_pred', fontsize = 20)
plt.xlabel('y_test', fontsize = 18)
plt.ylabel('y_test_pred', fontsize = 16)
plt.show()
```



Final Model Analysis

As per our final regression model, following features are the top predictors of bike sharing demand -

- **Temperature**: This is the strongest predictor of bike sharing demand with a positive coefficient of 0.4777. As the temperature rises, there is a very high probability of increase in demand.
- **Weather situation 3**(Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds): This kind of weather situation with a high negative coefficient of -0.2850 indicates a drop in bike sharing demand.
- **Year**: Again with a positive coefficient of 0.2341, this variable suggests that as the year passes with consistent presence of business, the service will become popular and hence demand would grow.

Thank You!