K L UNIVERSITY

COMPUTER SCIENCE ENGINEERING DEPARTMENT

A Project Based Lab Report

On

Artificial Intelligence for Data Science Applications

SUBMITTED BY:

ID NUMBER	NAME
2000030639	Mohammad Sameer

UNDER THE ESTEEMED GUIDANCE OF

Dr. Nilu Singh

Associate Professor

KL UNIVERSITY

Green Fields, Vaddeswaram – 522 502

Guntur Dt., AP, India.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

CERTIFICATE

ABSTRACT:

This report is about finding, working and implementing Regression on Capital Bike Sharing dataset

Build a system that recognizes English speech, using the DeepSpeech2 (DS2) model

AI and puzzle solver: Othello

Regression on Capital Bike Sharing dataset

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 system that recognizes English speech, using the DeepSpeech2 (DS2) model

We show that an end-to-end deep learning approach can be used to recognize either English or Mandarin Chinese speech--two vastly different languages. Because it replaces entire pipelines of hand-engineered components with neural networks, end-to-end learning allows us to handle a diverse variety of speech including noisy environments, accents and different languages. Key to our approach is our application of HPC techniques, resulting in a 7x speedup over our previous system. Because of this efficiency, experiments that previously took weeks now run in days. This enables us to iterate more quickly to identify superior architectures and algorithms. As a result, in several cases, our system is competitive with the transcription of human workers when benchmarked on standard datasets. Finally, using a technique called Batch Dispatch with GPUs in the data center, we show that our system can be inexpensively deployed in an online setting, delivering low latency when serving users at scale.

AI and puzzle solver: Othello

Operations research and management science are often confronted with sequential decision making problems with large state spaces. Standard methods that are used for solving such complex problems are associated with some difficulties. As we discuss in this article, these methods are plagued by the so-called curse of dimensionality and the curse of modelling. In this article, we discuss reinforcement learning, a machine learning technique for solving sequential decision making problems with large state spaces. We describe how reinforcement learning can be combined with a function approximation method to avoid both the curse of dimensionality and the curse of modelling. To illustrate the usefulness of this approach, we apply it to a problem with a huge state space—learning to play the game of Othello. We describe experiments in which reinforcement learning agents learn to play the game of Othello without the use of any knowledge provided by human experts. It turns out that the reinforcement learning agents learn to play the game of Othello better than players that use basic strategies.

INTRODUCTION TO PROBLEM STATEMENT:

Multiple Linear Regression

Bike Sharing Assignment

Problem Statement:

A US bike-sharing provider BikeIndia has recently suffered considerable dips in their revenues due to the ongoing Corona pandemic. The company is finding it very difficult to sustain in the current market scenario. So, it has decided to come up with a mindful business plan to be able to accelerate its revenue as soon as the ongoing lockdown comes to an end, and the economy restores to a healthy state.

In such an attempt, **BikeIndia** aspires to understand the demand for shared bikes among the people after this ongoing quarantine situation ends across the nation due to Covid-19. They have planned this to prepare themselves to cater to the people's needs once the situation gets better all around and stand out from other service providers and make huge profits.

They have contracted a consulting company 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:

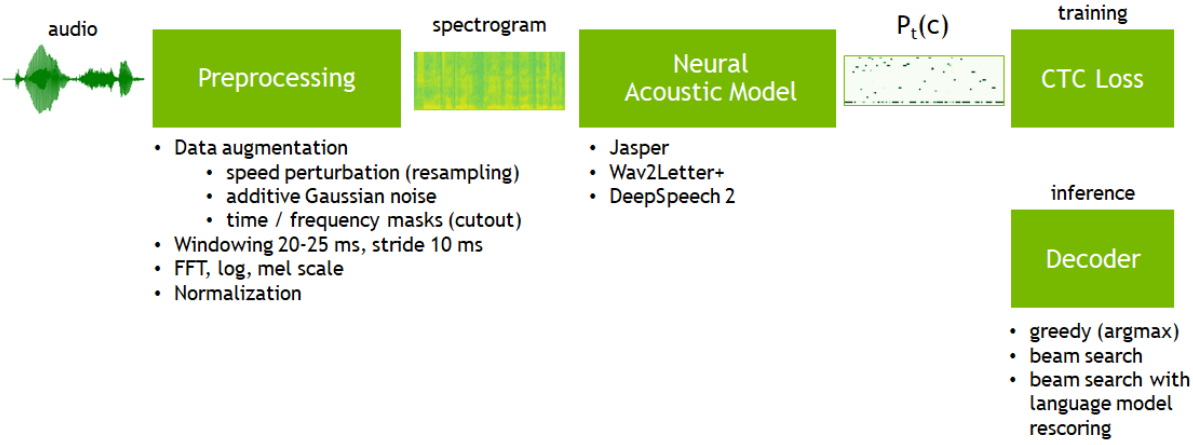
Which variables are significant in predicting the demand for shared bikes. How well those variables describe the bike demands Based on various meteorological surveys and people's styles, the service provider firm has gathered a large dataset on daily bike demands across the American market based on some factors.

Business Goal:
We are required to model the demand for shared bikes with the available independent variables. 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.

a system that recognizes English speech, using the DeepSpeech2 (DS2) model

Introduction

Automatic speech recognition (ASR) systems can be built using a number of approaches depending on input data type, intermediate representation, model's type and output post-processing. OpenSeq2Seq is currently focused on end-to-end CTC-based models (like original DeepSpeech model). These models are called end-to-end because they take speech samples and transcripts without any additional information. CTC allows finding an alignment between audio and text. CTC ASR models can be summarized in the following scheme:



Training pipeline consists of the following blocks:

1. audio preprocessing (feature extraction): signal normalization, windowing, (log) spectrogram (or mel scale spectrogram, or MFCC)

1. neural acoustic model (which predicts a probability distribution *P_t(c)* over vocabulary characters *c* per each time step *t* given input features per each t imestep)
2. CTC loss function

Inference pipeline is different for block #3:

3. decoder (which transforms a probability distribution into actual transcript)

We support different options for these steps. The recommended pipeline is the following (in order to get the best accuracy, the lowest WER):

1. Mel scale log spectrograms for audio features (using *librosa* backend)
2. Jasper as a neural acoustic model
3. Baidu’s CTC beam search decoder with N-gram language model rescoring

AI and puzzle solver: Othello

As far as board game searches/evaluation functions have come, I am not content with pruning the game tree of Othello. But, to make a brute force approach feasible, IMO we need an orders of magnitude speedup in processing the game tree. So, what's a man to do? I have been thinking along these lines: how can we transform Othello into a different, and possibly simpler, system that is more amenable to analysis

Theoretical Background :

Bike Sharing : Multiple Linear Regression

Removing redundant & unwanted columns

linkcode

Based on the high level look at the data and the data dictionary, the following variables can be removed from further analysis:

1. **instant** : Its only an index value
2. **dteday** : This has the date, Since we already have seperate columns for 'year' & 'month',hence, we could live without this column.
3. **casual & registered** : Both these columns contains the count of bike booked by different categories of customers. Since our objective is to find the total count of bikes and not by specific category, we will ignore these two columns. More over, we have created a new variable to have the ratio of these customer types.
4. We will save the new dataframe as bike_new, so that the original dataset is preserved for any future analysis/validation

Decoders

In order to get words out of a trained model one needs to use a decoder. Decoder converts a probability distribution over characters into text. There are two types of decoders that are usually employed with CTC-based models: greedy decoder and beam search decoder with language model re-scoring.

A greedy decoder outputs the most probable character at each time step. It is very fast and it can produce transcripts that are very close to the original pronunciation. But it may introduce many small misspelling errors. Due to the nature of WER metric, even one character error makes a whole word incorrect.

A beam search decoder with language model re-scoring allows checking many possible decodings (beams) at once with assigning a higher score for more probable N-grams according to a given language model.

AI and puzzle solver: Othello

One idea I've been toying with, in the above vein, is to represent the board differently (e.g. via polygons, vectors, lines etc), in the hope that one or more operations that deal with the board could be significantly optimized. Yesterday, when I asked my question on Quora.com, I thought of representing the board as a sequence of straight horizontal, vertical and diagonal <i>snakes</i>.

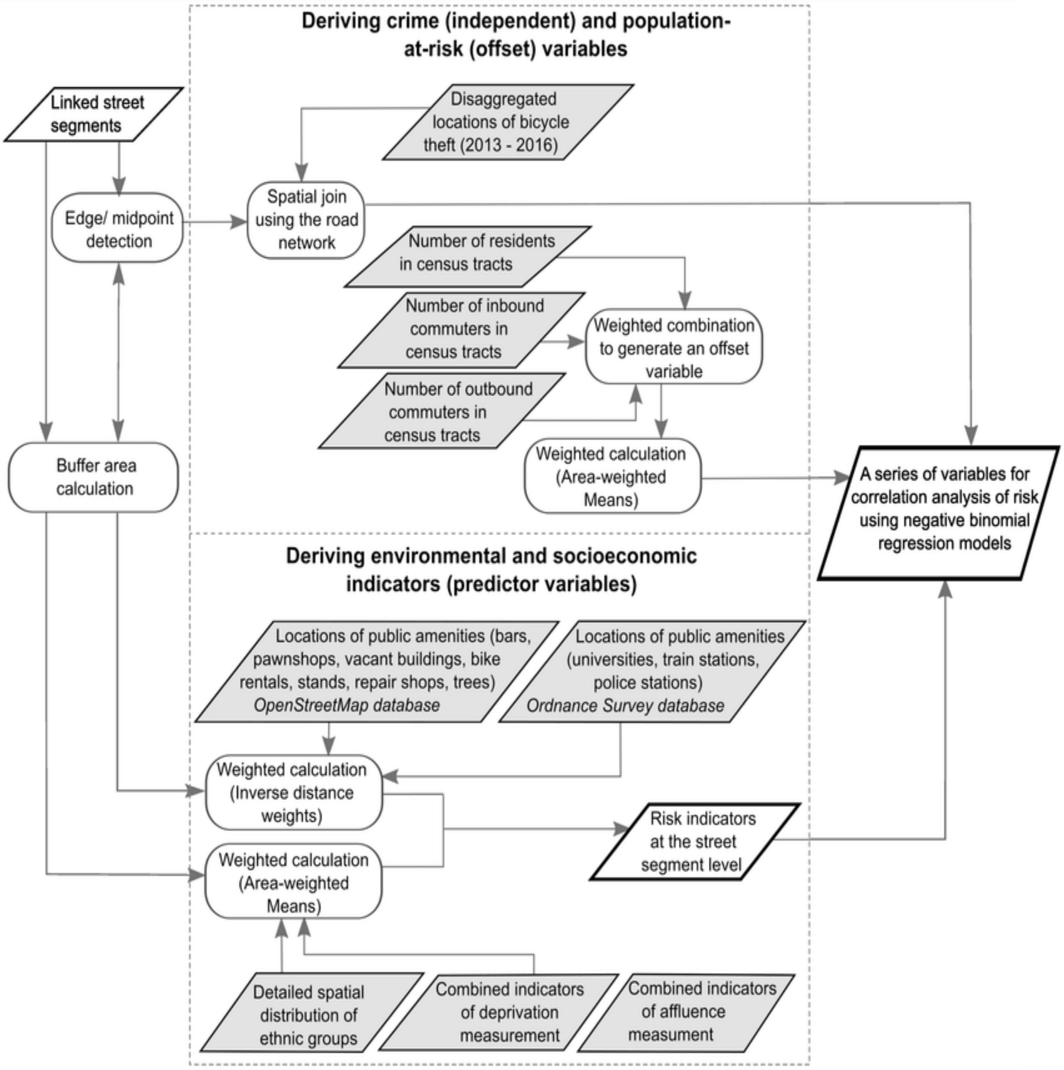
A snake is composed of one or more contiguous board squares, and has properties that can be accessed in constant time e.g. type, length, start, ending. At the start of the game, there would be six snakes [(x, y) below...]. Snake types are simple regular expressions e.g. _BAAAAAA_, AAAAAA, _BBBBBB.

Software Requirements:

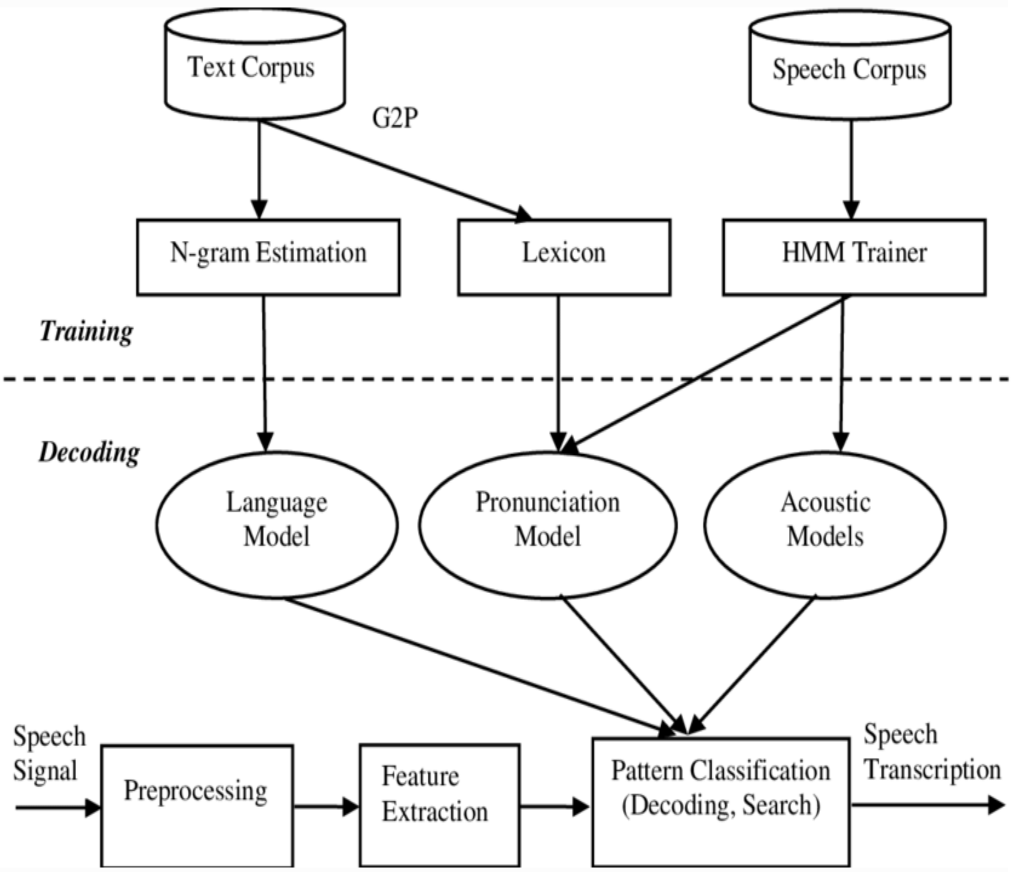
1) ANACONDA NAVIGATOR
2) JUPYTER NOTE BOOK
3) PYTHON IDE
4) RAPID MINER
5) ARIMA

Flow chart

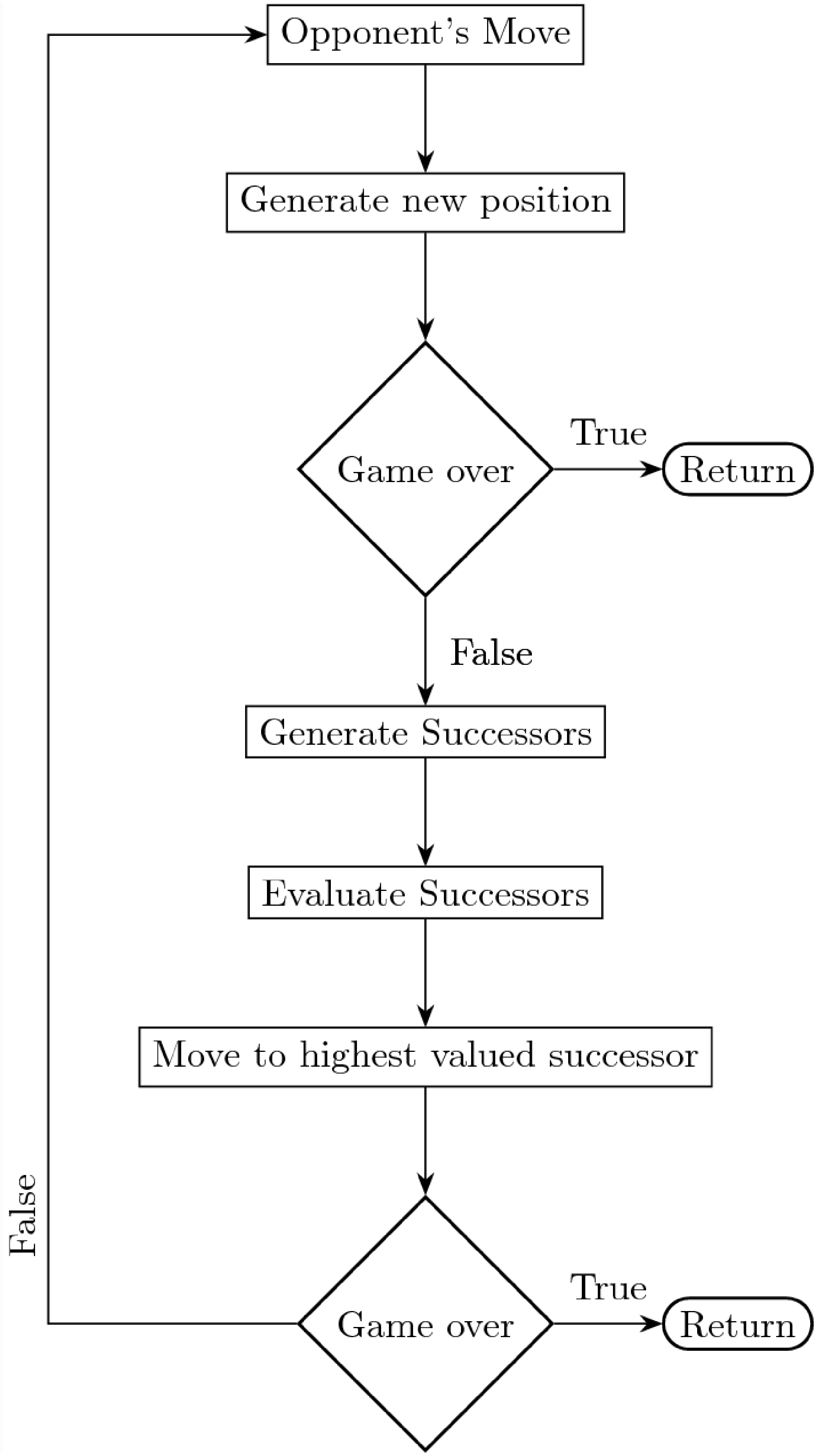
Regression on Capital Bike Sharing dataset



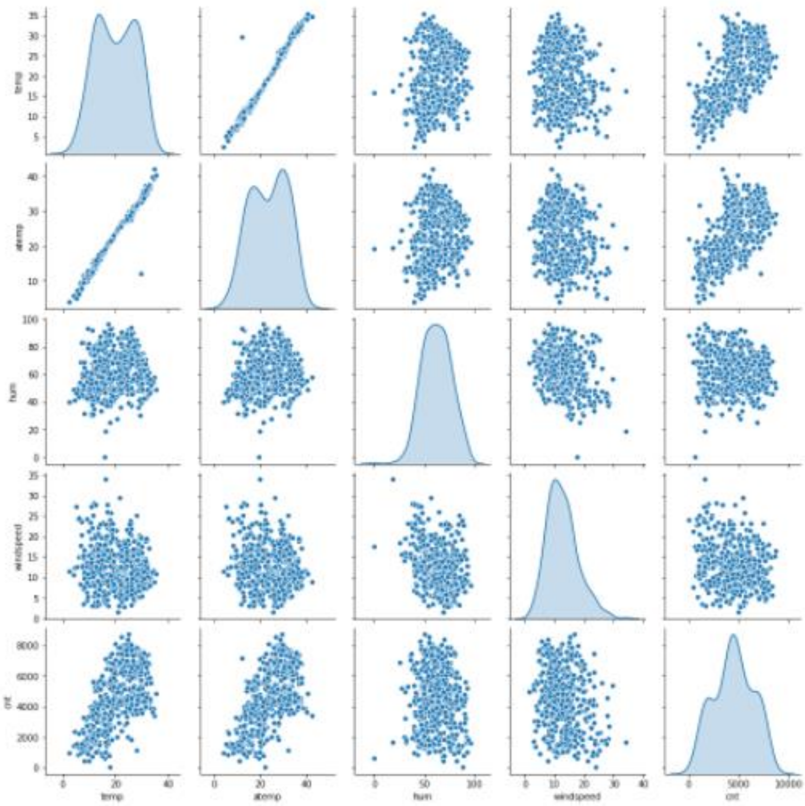
Build a system that recognizes English speech, using the DeepSpeech2 (DS2) model



AI and puzzle solver: Othello


Opponent's Move
Generate new position
Game over
True
Return
False
Generate Successors
Evaluate Successors
Move to highest valued successor
False
Game over
True
Return

Data Analytics: EDA and Plotting



Insights

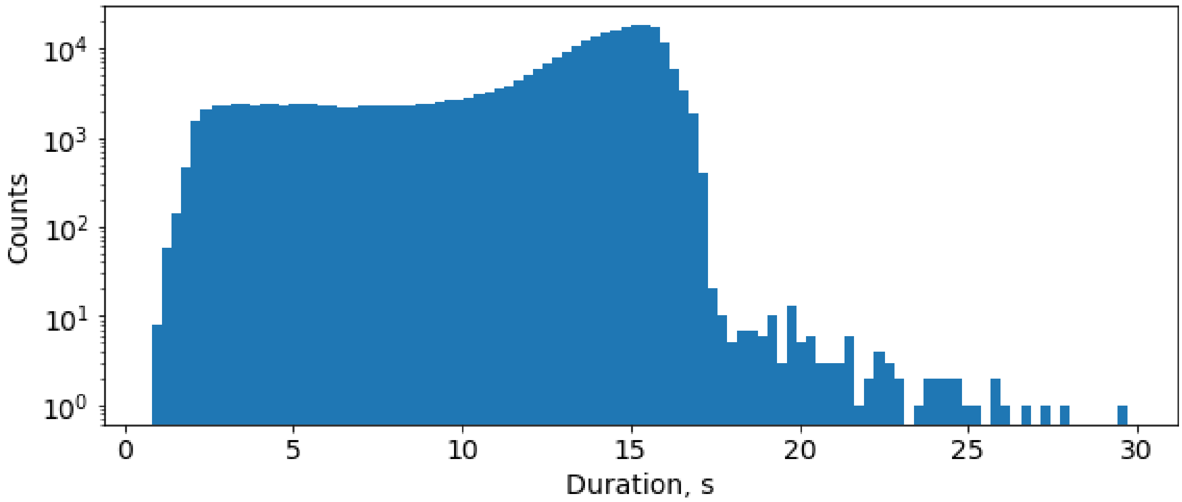
- The above Pair-Plot tells us that there is a LINEAR RELATION between 'temp','atemp' and 'cnt'

The model was trained with a Stochastic Gradient Descent with Momentum optimizer which was extended with Layer-wise Adaptive Rate Clipping (LARC) algorithm. The optimizer's parameters are the following:

- learning rate policy is polynomial with initial learning rate = 0.001 and power = 0.5
- momentum is 0.9
- batch size per GPU is 16

L2 weight decay (0.0005), dropout (0.5) and batch normalization were employed for regularization.

In training mode preprocessing augments original audio clips with additive noise and slight time stretching (to make speech faster/slower and increase/decrease its pitch). Duration of audio samples in LibriSpeech train dataset varies from 0.8 to 30 seconds:



Since 99.05% of the samples are shorter than 16.7 seconds, the preprocessing part ignores longer samples during the training. Such a filtering threshold can be set in the model's configuration file as "*max_duration*" parameter.

Coding

Multiple Linear Regression

Bike Sharing Assignment

Reading and Understanding the Data¶

In [1]:

```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

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

In [3]:

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
/kaggle/input/bike-sharing/day.csv
```

In [4]:

```
bike = pd.DataFrame(pd.read_csv("/kaggle/input/bike-sharing/day.csv"))
```

In [5]:

```
# Check the head of the dataset
bike.head()
```

Out[5]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	Temp	atemp	hum	windspeed	casual	registered	cnt
0	1	01-01-2018	1	0	1	0	6	0	2	14.110847	18.18125	80.5833	10.749882	331	654	985
1	2	02-01-2018	1	0	1	0	0	0	2	14.902598	17.68695	69.6087	16.652113	131	670	801
2	3	03-01-2018	1	0	1	0	1	1	1	8.050924	9.47025	43.7273	16.636703	120	1229	1349
3	4	04-01-2018	1	0	1	0	2	1	1	8.200000	10.60610	59.0435	10.739832	108	1454	1562
4	5	05-01-2018	1	0	1	0	3	1	1	9.305237	11.46350	43.6957	12.522300	82	1518	1600

In [6]:

```
# Check the descriptive information
bike.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 [7]:

```
bike.describe()
```

Out[7]:

	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	Temp	atemp	hum	windspeed	casual	registered	cnt
count	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	730.000000	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	0.683562	1.394521	20.319259	23.726322	62.765175	12.763620	849.249315	3658.757534	4508.006849
std	210.877136	1.110184	0.500343	3.450215	0.167266	2.006161	0.465405	0.544807	7.506729	8.150308	14.237589	5.195841	686.479875	1559.758728	1936.011647
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	2.424346	3.953480	0.000000	1.500244	2.000000	20.000000	22.000000
25%	183.250000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	13.811885	16.889713	52.000000	9.041650	316.250000	2502.250000	3169.750000

	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	Temp	atemp	hum	windspeed	casual	registered	cnt
50%	365.500000	3.000000	0.500000	7.000000	0.000000	3.000000	1.000000	1.000000	20.465826	24.368225	62.625000	12.125325	717.000000	3664.500000	4548.500000
75%	547.750000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000	26.880615	30.445775	72.989575	15.625589	1096.500000	4783.250000	5966.000000
max	730.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	35.328347	42.044800	97.250000	34.000021	3410.000000	6946.000000	8714.000000

In [8]:

```
# Check the shape of df

print(bike.shape)
(730, 16)
```

Finding :

Dataset has 730 rows and 16 columns.

Except one column, all other are either float or integer type.

One column is date type.

Looking at the data, there seems to be some fields that are categorical in nature, but in integer/float type.

We will analyse and finalize whether to convert them to categorical or treat as integer.

DATA QUALITY CHECK

Check for NULL/MISSING values

In [9]:

```
# percentage of missing values in each column
round(100*(bike.isnull().sum()/len(bike)), 2).sort_values(ascending=False)
```

Out[9]:

```
cnt           0.0
registered    0.0
```

```
casual        0.0
windspeed     0.0
hum           0.0
atemp         0.0
temp          0.0
weathersit    0.0
workingday    0.0
weekday       0.0
holiday       0.0
mnth          0.0
yr            0.0
season        0.0
dteday        0.0
instant       0.0
dtype: float64
```

In [10]:

```
# row-wise null count percentage
round((bike.isnull().sum(axis=1)/len(bike))*100,2).sort_values(ascending=False)
```

Out[10]:

```
729    0.0
250    0.0
248    0.0
247    0.0
246    0.0
      ... 
484    0.0
483    0.0
482    0.0
481    0.0
0      0.0
Length: 730, dtype: float64
```

Finding

There are no missing / Null values either in columns or rows

Duplicate Check

In [11]:

```
bike_dup = bike.copy()

# Checking for duplicates and dropping the entire duplicate row if any
bike_dup.drop_duplicates(subset=None, inplace=True)
```

In [12]:

```
bike_dup.shape
```

Out[12]:

```
(730, 16)
```

In [13]:

```
bike.shape
```

Out[13]:

```
(730, 16)
```

Insights

The shape after running the drop duplicate command is same as the original dataframe.

Hence we can conclude that there were zero duplicate values in the dataset.

Data Cleaning

Checking value_counts() for entire dataframe.

This will help to identify any Unknow/Junk values present in the dataset.

In [14]:

```
#Create a copy of the  dataframe, without the 'instant' column,

#as this will have unique values, and donot make sense to do a value count on it.

bike_dummy=bike.iloc[:,1:16]
```

In [15]:

```
for col in bike_dummy:
    print(bike_dummy[col].value_counts(ascending=False), '\n\n\n')
16-01-2019    1
10-09-2019    1
09-08-2018    1
25-09-2018    1
06-01-2018    1
             ..
06-04-2019    1
08-01-2018    1
27-02-2018    1
20-07-2019    1
03-05-2018    1
Name: dteday, Length: 730, dtype: int64



3    188
2    184
1    180
4    178
Name: season, dtype: int64



1    365
0    365
Name: yr, dtype: int64



12    62
10    62
8     62
7     62
5     62
```

```
3     62
1     62
11    60
9     60
6     60
4     60
2     56
Name: mnth, dtype: int64
```

```
0    709
1     21
Name: holiday, dtype: int64
```

```
6    105
1    105
0    105
5    104
4    104
2    104
3    103
Name: weekday, dtype: int64
```

```
1    499
0    231
Name: workingday, dtype: int64
```

```
1    463
2    246
3     21
Name: weathersit, dtype: int64
```

```
10.899153    5
26.035000    5
23.130847    4
28.563347    4
27.880000    4
            ..
27.025847    1
19.270000    1
13.191299    1
24.155847    1
5.526103     1
Name: temp, Length: 498, dtype: int64
```

```
32.73440    4
18.78105    3
31.85040    3
16.28750    2
17.58145    2
           ..
36.96315    1
24.93625    1
32.73460    1
14.82130    1
9.31250     1
Name: atemp, Length: 689, dtype: int64
```

```
61.3333    4
69.7083    3
59.0000    3
57.0000    3
72.9583    3
          ..
64.7917    1
44.9583    1
71.2083    1
50.0417    1
49.8750    1
Name: hum, Length: 594, dtype: int64
```

```
7.416900     3
15.333486    3
7.959064     3
11.166689    3
7.125450     3
            ..
14.500475    1
8.250514     1
19.416332    1
16.522200    1
9.750175     1
Name: windspeed, Length: 649, dtype: int64
```

```
968     4
120     4
639     3
653     3
163     3
       ..
1639    1
616     1
620     1
1278    1
1488    1
```

```
Name: casual, Length: 605, dtype: int64
```

```
1707    3
6248    3
4841    3
3461    2
5265    2
       ..
2720    1
670     1
1693    1
4763    1
4097    1
Name: registered, Length: 678, dtype: int64
```

```
5119    2
4274    2
3784    2
6883    2
2077    2
       ..
6273    1
5501    1
4760    1
1683    1
4097    1
Name: cnt, Length: 695, dtype: int64
```

Insights

There seems to be no Junk/Unknown values in the entire dataset.

```
workingday      0.040787
temp            0.433878
atemp           0.058635
hum            -0.178382
windspeed      -0.184925
season_2        0.130228
season_3        0.079599
season_4        0.153475
mnth_3          0.047149
mnth_9          0.100017
mnth_10         0.054370
weekday_6       0.054618
weathersit_2   -0.047472
weathersit_3   -0.271174
dtype: float64
```

In [54]:

```
# Print a summary of the linear regression model obtained
print(lr1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.842
Model:                            OLS   Adj. R-squared:                  0.837
Method:                 Least Squares   F-statistic:                     175.1
Date:                Sat, 22 Jan 2022   Prob (F-statistic):          1.28e-186
Time:                        13:16:48   Log-Likelihood:                 509.26
No. Observations:                 510   AIC:                            -986.5
Df Residuals:                     494   BIC:                            -918.8
Df Model:                          15
Covariance Type:            nonrobust
==============================================================================
==
                   coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const            0.1953      0.030      6.576      0.000       0.137       0.2
54
yr               0.2287      0.008     28.013      0.000       0.213       0.2
45
workingday       0.0408      0.011      3.705      0.000       0.019       0.0
62
temp             0.4339      0.134      3.238      0.001       0.171       0.6
97
atemp            0.0586      0.137      0.427      0.670      -0.211       0.3
28
hum             -0.1784      0.037     -4.777      0.000      -0.252      -0.1
05
windspeed       -0.1849      0.028     -6.612      0.000      -0.240      -0.1
30
season_2         0.1302      0.015      8.575      0.000       0.100       0.1
60
season_3         0.0796      0.021      3.818      0.000       0.039       0.1
21
season_4         0.1535      0.014     10.765      0.000       0.125       0.1
81
mnth_3           0.0471      0.016      2.958      0.003       0.016       0.0
78
mnth_9           0.1000      0.016      6.303      0.000       0.069       0.1
31
mnth_10          0.0544      0.018      3.046      0.002       0.019       0.0
89
weekday_6        0.0546      0.014      3.818      0.000       0.027       0.0
83
weathersit_2    -0.0475      0.011     -4.455      0.000      -0.068      -0.0
27
weathersit_3    -0.2712      0.028     -9.542      0.000      -0.327      -0.2
15
==============================================================================
Omnibus:                       92.576   Durbin-Watson:                   2.037
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              221.202
Skew:                          -0.933   Prob(JB):                     9.26e-49
Kurtosis:                       5.632   Cond. No.                         85.8
==============================================================================
```

```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
```

Model 2

- Removing the variable 'atemp' based on its High p-value & High VIF

In [55]:

```python
X_train_new = X_train_rfe.drop(["atemp"], axis = 1)
```

VIF Check

In [56]:

```python
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Create a dataframe that will contain the names of all the feature variables and
their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train_new.columns
vif['VIF'] = [variance_inflation_factor(X_train_new.values, i) for i in range(X_tr
ain_new.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[56]:

	Features	VIF
2	temp	23.21
3	hum	17.23
6	season_3	7.01
1	workingday	4.60
4	windspeed	4.55
5	season_2	3.54

	Features	VIF
7	season_4	3.01
12	weathersit_2	2.14
0	yr	2.02
11	weekday_6	1.79
10	mnth_10	1.66
9	mnth_9	1.28
8	mnth_3	1.20
13	weathersit_3	1.17

In [57]:

```
# Add a constant
X_train_lm2 = sm.add_constant(X_train_new)

# Create a first fitted model
lr2 = sm.OLS(y_train, X_train_lm2).fit()
```

In [58]:

```
# Check the parameters obtained

lr2.params
```

Out[58]:

```
const           0.196221
yr              0.228723
workingday      0.040773
temp            0.489280
hum            -0.177805
windspeed      -0.187198
season_2        0.130352
season_3        0.078664
```

```
season_4        0.153732
mnth_3          0.047295
mnth_9          0.100029
mnth_10         0.054438
weekday_6       0.054705
weathersit_2   -0.047620
weathersit_3   -0.271535
dtype: float64
```

In [59]:

```
# Print a summary of the linear regression model obtained
print(lr2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.842
Model:                            OLS   Adj. R-squared:                  0.837
Method:                 Least Squares   F-statistic:                     187.9
Date:                Sat, 22 Jan 2022   Prob (F-statistic):         1.00e-187
Time:                        13:16:51   Log-Likelihood:                 509.17
No. Observations:                 510   AIC:                            -988.3
Df Residuals:                     495   BIC:                            -924.8
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
==
                   coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const            0.1962      0.030      6.627      0.000       0.138       0.2
54
yr               0.2287      0.008     28.034      0.000       0.213       0.2
45
workingday       0.0408      0.011      3.706      0.000       0.019       0.0
62
temp             0.4893      0.034     14.595      0.000       0.423       0.5
55
hum             -0.1778      0.037     -4.769      0.000      -0.251      -0.1
05
windspeed       -0.1872      0.027     -6.823      0.000      -0.241      -0.1
33
season_2         0.1304      0.015      8.592      0.000       0.101       0.1
60
season_3         0.0787      0.021      3.797      0.000       0.038       0.1
19
season_4         0.1537      0.014     10.802      0.000       0.126       0.1
82
mnth_3           0.0473      0.016      2.971      0.003       0.016       0.0
79
mnth_9           0.1000      0.016      6.309      0.000       0.069       0.1
31
mnth_10          0.0544      0.018      3.052      0.002       0.019       0.0
89
weekday_6        0.0547      0.014      3.828      0.000       0.027       0.0
83
```

```
weathersit_2    -0.0476      0.011     -4.475      0.000     -0.069      -0.0
27
weathersit_3    -0.2715      0.028     -9.567      0.000     -0.327      -0.2
16
==============================================================================
Omnibus:                       92.002   Durbin-Watson:                   2.038
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              219.387
Skew:                          -0.929   Prob(JB):                     2.29e-48
Kurtosis:                       5.622   Cond. No.                         21.2
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
```

Model 3

- Removing the variable 'hum' based on its Very High 'VIF' value.
- Even though the VIF of hum is second highest, we decided to drop 'hum' and not 'temp' based on general knowledge that temperature can be an important factor for a business like bike rentals, and wanted to retain 'temp'.

In [60]:

```
X_train_new = X_train_new.drop(["hum"], axis = 1)
```

VIF Check

In [61]:

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Create a dataframe that will contain the names of all the feature variables and
their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train_new.columns
vif['VIF'] = [variance_inflation_factor(X_train_new.values, i) for i in range(X_tr
ain_new.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[61]:

	Features	VIF
2	temp	16.81
5	season_3	6.75
3	windspeed	4.27

	Features	VIF
1	workingday	4.11
4	season_2	3.51
6	season_4	2.89
0	yr	2.02
9	mnth_10	1.66
10	weekday_6	1.66
11	weathersit_2	1.54
8	mnth_9	1.27
7	mnth_3	1.20
12	weathersit_3	1.08

In [62]:

```
# Add a constant
X_train_lm3 = sm.add_constant(X_train_new)

# Create a first fitted model
lr3 = sm.OLS(y_train, X_train_lm3).fit()
```

In [63]:

```
lr3.params
```

Out[63]:

```
const          0.091594
yr             0.233129
```

```
workingday      0.042443
temp            0.456709
windspeed      -0.148815
season_2        0.131914
season_3        0.087922
season_4        0.150243
mnth_3          0.055303
mnth_9          0.091371
mnth_10         0.053320
weekday_6       0.055451
weathersit_2   -0.077149
weathersit_3   -0.324223
dtype: float64
```

In [64]:

```
# Print a summary of the linear regression model obtained
print(lr3.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.834
Model:                            OLS   Adj. R-squared:                  0.830
Method:                 Least Squares   F-statistic:                     192.2
Date:                Sat, 22 Jan 2022   Prob (F-statistic):          4.52e-184
Time:                        13:16:53   Log-Likelihood:                 497.71
No. Observations:                 510   AIC:                            -967.4
Df Residuals:                     496   BIC:                            -908.1
Df Model:                          13
Covariance Type:            nonrobust
==============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const          0.0916      0.020      4.509      0.000       0.052       0.1
32
yr             0.2331      0.008     28.149      0.000       0.217       0.2
49
workingday     0.0424      0.011      3.778      0.000       0.020       0.0
65
temp           0.4567      0.034     13.620      0.000       0.391       0.5
23
windspeed     -0.1488      0.027     -5.553      0.000      -0.201      -0.0
96
season_2       0.1319      0.015      8.512      0.000       0.101       0.1
62
season_3       0.0879      0.021      4.172      0.000       0.047       0.1
29
season_4       0.1502      0.015     10.346      0.000       0.122       0.1
79
mnth_3         0.0553      0.016      3.419      0.001       0.024       0.0
87
mnth_9         0.0914      0.016      5.678      0.000       0.060       0.1
23
mnth_10        0.0533      0.018      2.926      0.004       0.018       0.0
89
```

```
weekday_6        0.0555      0.015      3.798      0.000       0.027       0.0
84
weathersit_2    -0.0771      0.009     -8.727      0.000      -0.095      -0.0
60
weathersit_3    -0.3242      0.027    -12.139      0.000      -0.377      -0.2
72
==============================================================================
Omnibus:                       87.519   Durbin-Watson:                   2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              205.489
Skew:                          -0.891   Prob(JB):                     2.39e-45
Kurtosis:                       5.548   Cond. No.                         16.3
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
```

Model 4

- Removing the variable 'season3' based on its Very High 'VIF' value.
- Even though the VIF of season3 is second highest, we decided to drop 'season3' and not 'temp' based on general knowledge that temperature can be an important factor for a business like bike rentals, and wanted to retain 'temp'.

In [65]:

```
X_train_new = X_train_new.drop(["season_3"], axis = 1)
```

VIF Check

In [66]:

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Create a dataframe that will contain the names of all the feature variables and
their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train_new.columns
vif['VIF'] = [variance_inflation_factor(X_train_new.values, i) for i in range(X_tr
ain_new.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[66]:

	Features	VIF
2	temp	4.92
3	windspeed	4.15

	Features	VIF
1	workingday	4.07
0	yr	2.01
5	season_4	1.98
9	weekday_6	1.66
8	mnth_10	1.63
4	season_2	1.56
10	weathersit_2	1.54
7	mnth_9	1.23
6	mnth_3	1.15
11	weathersit_3	1.08

In [67]:

```
# Add a constant
X_train_lm4 = sm.add_constant(X_train_new)

# Create a first fitted model
lr4 = sm.OLS(y_train, X_train_lm4).fit()
```

In [68]:

```
# Check the parameters obtained

lr4.params
```

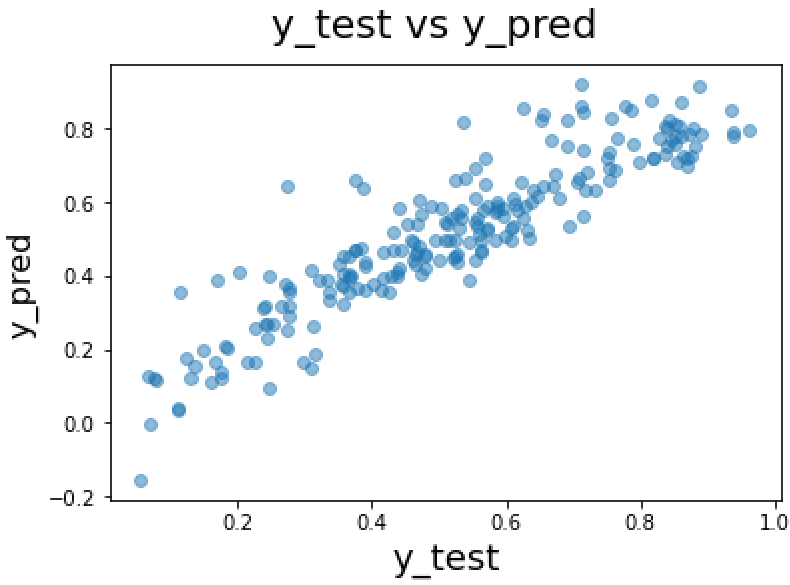
Out[68]:

MODEL EVALUATION

In [90]:

```
# Plotting y_test and y_pred to understand the spread

fig = plt.figure()
plt.scatter(y_test, y_pred, alpha=.5)
fig.suptitle('y_test vs y_pred', fontsize = 20)              # Plot heading
plt.xlabel('y_test', fontsize = 18)                          # X-label
plt.ylabel('y_pred', fontsize = 16)
plt.show()
```



Final Result Comparison

- Train R^2 :0.824
- Train Adjusted R^2 :0.821
- Test R^2 :0.820
- Test Adjusted R^2 :0.812
- This seems to be a really good model that can very well 'Generalize' various datasets.

FINAL REPORT

As per our final Model, the top 3 predictor variables that influences the bike booking are:

- **Temperature (temp)** - A coefficient value of '0.5636' indicated that a unit increase in temp variable increases the bike hire numbers by 0.5636 units.
- **Weather Situation 3 (weathersit_3)** - A coefficient value of '-0.3070' indicated that, w.r.t Weathersit1, a unit increase in Weathersit3 variable decreases the bike hire numbers by 0.3070 units.
- **Year (yr)** - A coefficient value of '0.2308' indicated that a unit increase in yr variable increases the bike hire numbers by 0.2308 units.

So, it's suggested to consider these variables utmost importance while planning, to achive maximum Booking

The next best features that can also be considered are

- **season_4:** - A coefficient value of '0.128744' indicated that w.r.t season_1, a unit increase in season_4 variable increases the bike hire numbers by 0.128744 units.
- **windspeed:** - A coefficient value of '-0.155191' indicated that, a unit increase in windspeed variable decreases the bike hire numbers by 0.155191 units.

  #### NOTE:

  - The details of weathersit_1 & weathersit_3
  - **weathersit_1:** Clear, Few clouds, Partly cloudy, Partly cloudy
  - **weathersit_3:** Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

The details of season1 & season4

- **season1:** spring
- **season4:** winter