Demand Forecasting

Can you forecast the demand of the car rentals on an hourly basis?

Problem Statement

ABC is a car rental company based out of Bangalore. It rents cars for both in and out stations at affordable prices. The users can rent different types of cars like Sedans, Hatchbacks, SUVs and MUVs, Minivans and so on.

In recent times, the demand for cars is on the rise. As a result, the company would like to tackle the problem of supply and demand. The ultimate goal of the company is to strike the balance between the supply and demand inorder to meet the user expectations.

The company has collected the details of each rental. Based on the past data, the company would like to forecast the demand of car rentals on an hourly basis.

Objective

The main objective of the problem is to develop the machine learning approach to forecast the demand of car rentals on an hourly basis.

Data Dictionary

You are provided with 3 files - train.csv, test.csv and sample_submission.csv

Training set

train.csv contains the hourly demand of car rentals from August 2018 to February 2021.

Variable	Description
date	Date (yyyy-mm-dd)
hour	Hour of the day
demand	No. of car rentals in a hour

Test set

test.csv contains only 2 variables: date and hour. You need to predict the hourly demand of car rentals for the next 1 year i.e. from March 2021 to March 2022.

Variable	Description
date	Date (yyyy-mm-dd)
hour	Hour of the day

Submission File Format

sample_submission.csv contains 3 variables - date, hour and demand

Variable	Description
date	Date (yyyy-mm-dd)
hour	Hour of the day
demand	No. of car rentals in a hour

Evaluation metric

The evaluation metric for this hackathon is RMSE score.

Solution:

In [1]:
```
#import libraries

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

```
<frozen importlib._bootstrap>:228: RuntimeWarning: scipy._lib.messagestream.MessageStrea
m size changed, may indicate binary incompatibility. Expected 56 from C header, got 64 f
rom PyObject
```

In [2]:
```
"""
check_holiday(date)

returns 1 if date is a holiday else 0
"""
def check_hoilday(date):
```

```
    if date in holidays.IN() or date in holidays.US():
        return 1
    else:
        return 0


"""
check_season(month)
0: summer
1: fall
2: winter

"""
def check_season(month):
    if month in [3,4,5,6]:
        return 0 #'summer'
    elif month in [7,8,9,10]:
        return 1#'fall'
    elif month in [11,12,1,2]:
        return 2 #'winter'
```

Feature Engineering

date -> is converted into `year` , `month` , `day` , `dayofweek` , and `season` # checking statistics of the data train_df.describe() columns. And Also `is_weekend` column is created for checking whether it is weekend or not

In [3]:
```
"""
Feature Engineering of the data

preprocessing(df)

"""
def preprocessing(df):
    df['date'] = pd.to_datetime(df['date'])
    df['year'] = df['date'].dt.year
    df['month'] = df['date'].dt.month
    df['day'] = df['date'].dt.day
    df['dayofweek'] = df['date'].dt.dayofweek
    df['season'] = df['month'].apply(check_season)
    df['is_weekend'] = df['dayofweek'].apply(lambda x: 1 if x in ['Sunday', 'Saturday',5
    df['after_weekend'] = df['dayofweek'].apply(lambda x: 1 if x == 'Monday' or x == 0 e
    df['before_weekend'] = df['dayofweek'].apply(lambda x: 1 if x == 'Saturday' or x ==
    df['is_holiday'] = df['date'].apply(check_hoilday)

    df.set_index('date', inplace=True)
    df.sort_index(inplace=True)
    return df
```

In [4]:
```
train_df = preprocessing(pd.read_csv('train_E1GspfA.csv')) # preprocessing the train dat
test_df = preprocessing(pd.read_csv('test_6QvDdzb.csv')) # preprocessing the test data
```

In [5]:
```
train_df.info() # check the dataframe and its columns, And also null values
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 18247 entries, 2018-08-18 to 2021-02-28
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   hour            18247 non-null  int64
 1   demand          18247 non-null  int64
 2   year            18247 non-null  int64
```

```
 3   month           18247 non-null  int64
 4   day             18247 non-null  int64
 5   dayofweek       18247 non-null  int64
 6   season          18247 non-null  int64
 7   is_weekend      18247 non-null  int64
 8   after_weekend   18247 non-null  int64
 9   before_weekend  18247 non-null  int64
 10  is_holiday      18247 non-null  int64
dtypes: int64(11)
memory usage: 1.7 MB
```

In [6]:
```
# checking statistics of the data
train_df.describe()
```

Out[6]:

	hour	demand	year	month	day	dayofweek	season
count	18247.000000	18247.000000	18247.000000	18247.000000	18247.000000	18247.000000	18247.000000
mean	12.614731	73.991451	2019.396997	6.470324	15.782430	3.025867	1.099414
std	6.544963	41.678988	0.810979	3.618189	8.772904	2.003638	0.812515
min	0.000000	1.000000	2018.000000	1.000000	1.000000	0.000000	0.000000
25%	8.000000	43.000000	2019.000000	3.000000	8.000000	1.000000	0.000000
50%	13.000000	71.000000	2019.000000	7.000000	16.000000	3.000000	1.000000
75%	18.000000	98.000000	2020.000000	10.000000	23.000000	5.000000	2.000000
max	23.000000	379.000000	2021.000000	12.000000	31.000000	6.000000	2.000000

EDA

From Dataset, we observerd that there are two independent variables(i.e., `date` , `hour`) but both are measuring time. and another variable is `demand` which is our target/ dependent variable.

Taking this into note, we devired new features from existing `date` feature like `year` , `month` , `relative date` , `day` , `season` , `is_weekend` , `after_weekend` , `before_weekend` , `is_hoilday`

`year` -> 2019,2020..
`month` -> Jan, Feb, March..... etc in numerial form
`day` -> Monday, Tuesday..etc in numerial form, 0-6
`season` -> summer, Rainy , winter

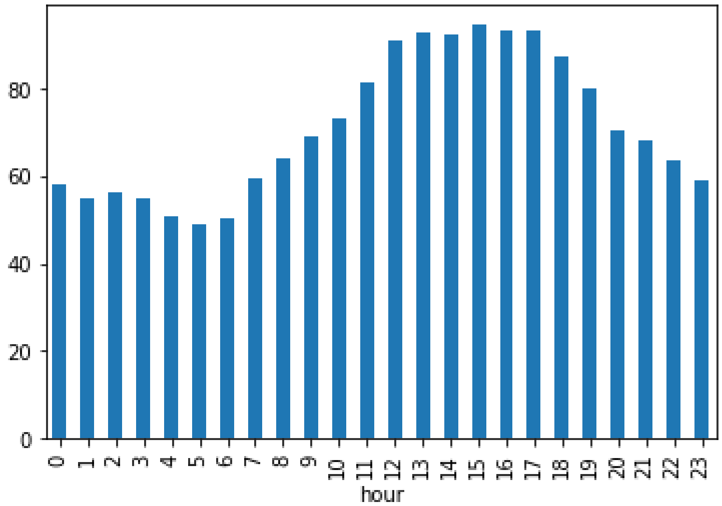
Average of the car rentals demand for each hour of the day

In [7]:
```
"""
Average of the car rentals demand for each hour of the day

"""
train_df.groupby('hour').mean()['demand'].plot(kind='bar')

"""""
Below graph shows the average demand for each hour of the day,  we conclude that the dem
rentals are high in between the hours of 11 to 20. and rest of the hours are low.
"""
```

Out[7]:
```
'" \nBelow graph shows the average demand for each hour of the day,  we conclude that th
e demand of cars \nrentals are high in between the hours of 11 to 20. and rest of the ho
urs are low.\n'
```



Average of the car rentals demand for each day of the month

In [8]:

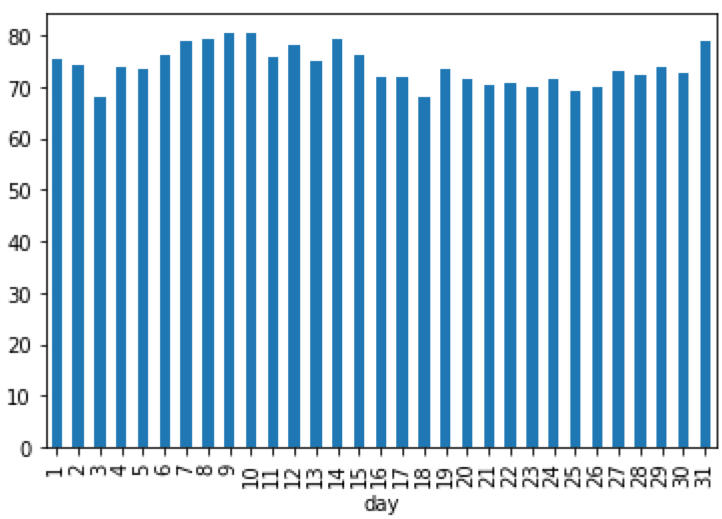
```
"""
Average of the car rentals demand for each day of the month

"""
train_df.groupby('day').mean()['demand'].plot(kind='bar')

""""
Below graph shows the average demand for each day of the month,  we conclude that the de
rentals are almost average upto 75.
"""
```

Out[8]:

```
'" \nBelow graph shows the average demand for each day of the month,  we conclude that t
he demand of cars \nrentals are almost average upto 75.\n'
```



Average of the car rentals demand for each day of the week

In [9]:

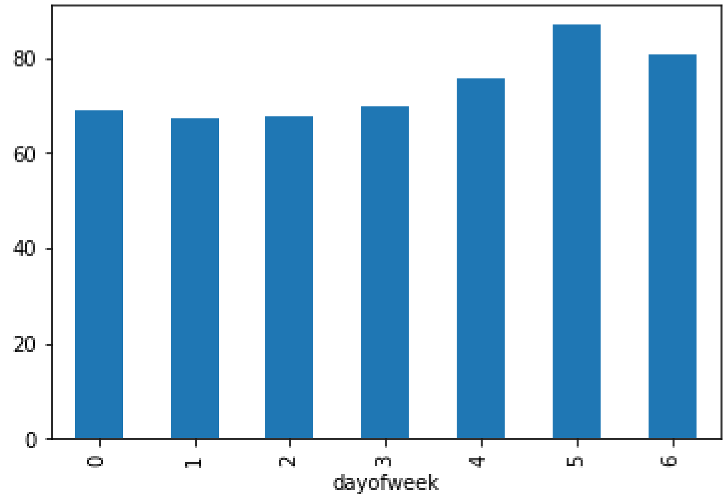
```
"""
Average of the car rentals demand for each day of the week

"""
train_df.groupby('dayofweek').mean()['demand'].plot(kind='bar')

"""
Below graph shows the average demand for each day of the week,  we conclude that the ave
for the end of week is higher than the remaning days of week.
"""
```

Out[9]:

```
' \nBelow graph shows the average demand for each day of the week,  we conclude that the
average demand \nfor the end of week is higher than the remaning days of week.\n'
```



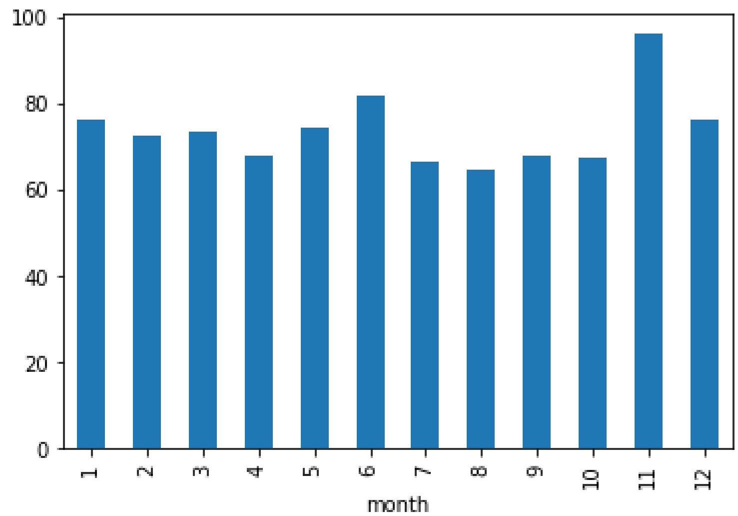
Average of the car rentals demand for each day of the week

In [10]:
```
"""
Average of the car rentals demand for each month of the year

"""
train_df.groupby('month').mean()['demand'].plot(kind='bar')

"""""
Below graph shows the average demand for each hour of the day,  we conclude that the dem
rentals are almost average upto 79. In November, the demand is higher than the rest of t
"""
```

Out[10]:
```
'" \nBelow graph shows the average demand for each hour of the day,  we conclude that th
e demand of cars \nrentals are almost average upto 79. In November, the demand is higher
than the rest of the months.\n'
```

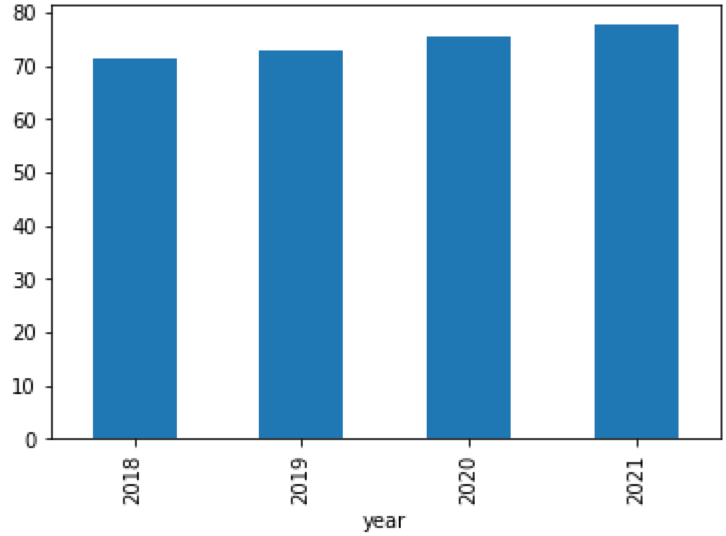


Average of the car rentals demand for each year

In [11]:
```
"""
Average of the car rentals demand for each year
"""
train_df.groupby('year').mean()['demand'].plot(kind='bar')

"""""
Below graph shows the average demand in years,  we conclude that the demand of cars
rentals are slightly increasing demand over the years.
"""
```

Out[11]:
```
'" \nBelow graph shows the average demand in years,  we conclude that the demand of cars
\nrentals are slightly increasing demand over the years.\n'
```



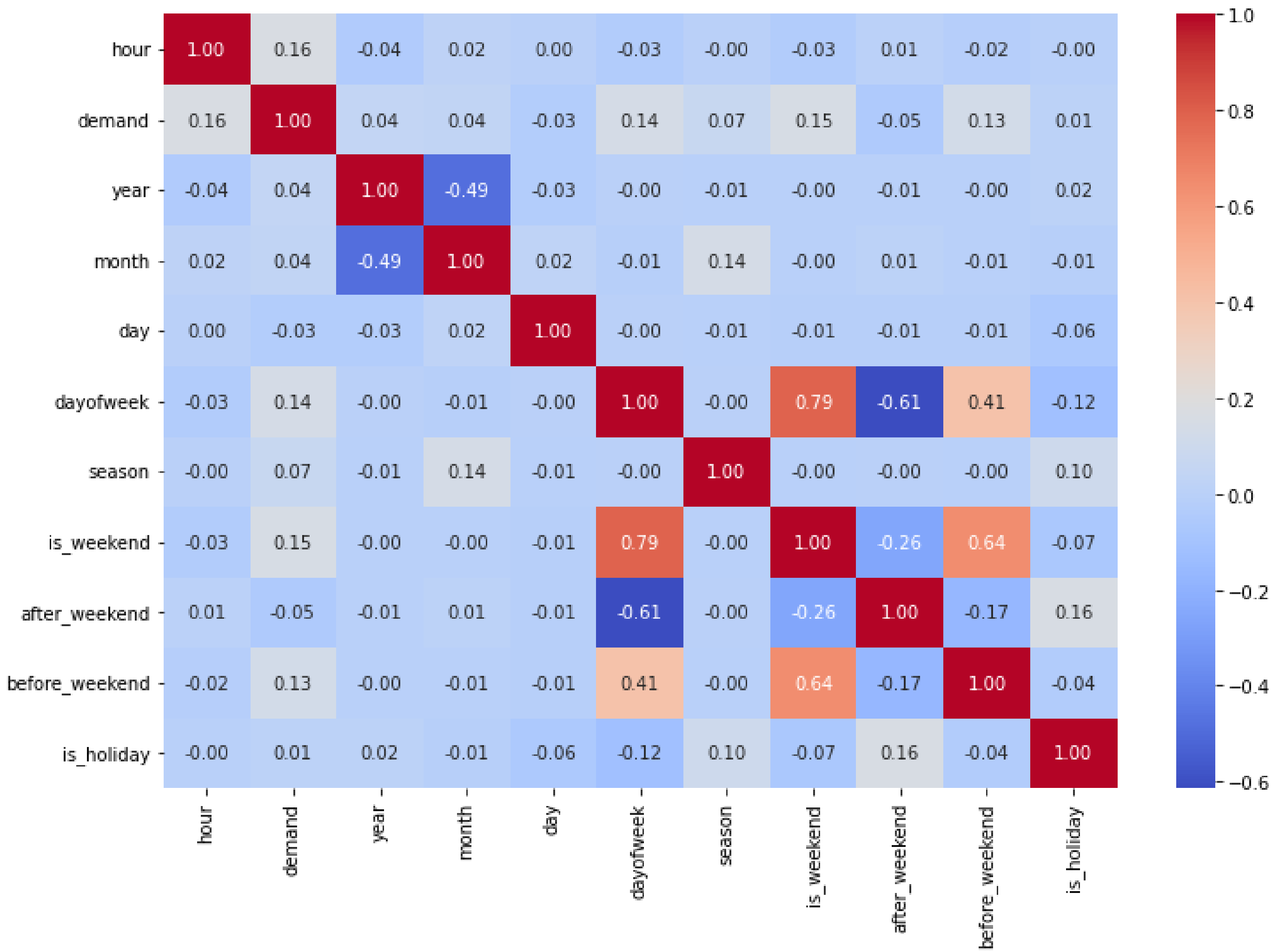
Correlation of Features on one to one bias:

In [12]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))

sns.heatmap(
train_df.corr(), fmt='.2f', annot=True, cmap='coolwarm')
```

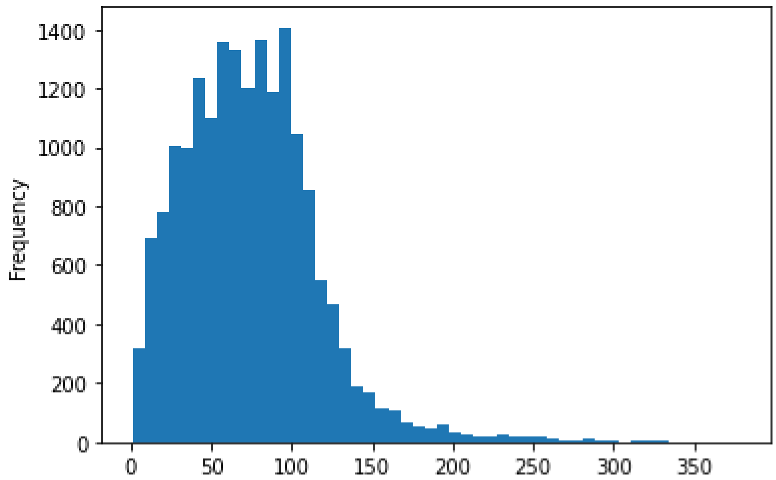
Out[12]: <AxesSubplot:>



	hour	demand	year	month	day	dayofweek	season	is_weekend	after_weekend	before_weekend	is_holiday
hour	1.00	0.16	-0.04	0.02	0.00	-0.03	-0.00	-0.03	0.01	-0.02	-0.00
demand	0.16	1.00	0.04	0.04	-0.03	0.14	0.07	0.15	-0.05	0.13	0.01
year	-0.04	0.04	1.00	-0.49	-0.03	-0.00	-0.01	-0.00	-0.01	-0.00	0.02
month	0.02	0.04	-0.49	1.00	0.02	-0.01	0.14	-0.00	0.01	-0.01	-0.01
day	0.00	-0.03	-0.03	0.02	1.00	-0.00	-0.01	-0.01	-0.01	-0.01	-0.06
dayofweek	-0.03	0.14	-0.00	-0.01	-0.00	1.00	-0.00	0.79	-0.61	0.41	-0.12
season	-0.00	0.07	-0.01	0.14	-0.01	-0.00	1.00	-0.00	-0.00	-0.00	0.10
is_weekend	-0.03	0.15	-0.00	-0.00	-0.01	0.79	-0.00	1.00	-0.26	0.64	-0.07
after_weekend	0.01	-0.05	-0.01	0.01	-0.01	-0.61	-0.00	-0.26	1.00	-0.17	0.16
before_weekend	-0.02	0.13	-0.00	-0.01	-0.01	0.41	-0.00	0.64	-0.17	1.00	-0.04
is_holiday	-0.00	0.01	0.02	-0.01	-0.06	-0.12	0.10	-0.07	0.16	-0.04	1.00

In [13]:
```
train_df['demand'].plot(kind='hist', bins=50) #Positive skewed distribution
```

Out[13]: <AxesSubplot:ylabel='Frequency'>



In [14]:
```
train_df.fillna(0, inplace=True) # null values replace with 0 bcoz, in hoilday column we
test_df.fillna(0, inplace=True) # null values replace with 0 bcoz, in hoilday column we
```

OneHot Encoding

In [15]:
```
#only onehot encoding -> dayofweek

train_df = pd.concat([train_df, pd.get_dummies(train_df['dayofweek'], prefix='dayofweek'
train_df.drop(['dayofweek'], axis=1, inplace=True)


test_df = pd.concat([test_df, pd.get_dummies(test_df['dayofweek'], prefix='dayofweek')],
test_df.drop(['dayofweek'], axis=1, inplace=True)
```

Spliting of dataset into Train and Test data

In [16]:
```
split = int(round(train_df.shape[0]*0.75, 0)) # 75 % trainset and 25 % testset split val

Xtrain = train_df[:split].drop(['demand'], axis=1)
ytrain = train_df[:split]['demand']

Xtest = train_df[split:].drop(['demand'], axis=1)
ytest = train_df[split:]['demand']
```

Model Training:

By Statistcal Analysis of Dataset, we found that it is a time-series problem but if we use time-series models, the predications are not good. Because `demand` variable is almost uniform distrubution. Finally, we concluded that the problem statement is a regression model in supervised learning to acheive better predications.

Linear Regression

In [17]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lr = LinearRegression()
lr.fit(Xtrain, ytrain)

ypred = lr.predict(Xtest)
print("RMSE Score:- {0}".format(mean_squared_error(ytest, ypred, squared=False)))
```

```
RMSE Score:- 41.94499989178477
```

RandomForest Regressor

In [18]:
```
from sklearn.ensemble import RandomForestRegressor

rnfr = RandomForestRegressor()
rnfr.fit(Xtrain, ytrain)

rnf_ypred = rnfr.predict(Xtest)
print("RMSE Score:- {0}".format(mean_squared_error(ytest, rnf_ypred, squared=False)))
```

```
RMSE Score:- 38.249143921960176
```

XGBoost:-

In [19]:
```
import xgboost as xgb

xgbr = xgb.XGBRegressor(learning_rate=0.05, objective='reg:squarederror', n_estimators=5
xgbr.fit(Xtrain, ytrain)

xgbrp = xgbr.predict(Xtest)
print("RMSE Score:- {0}".format(mean_squared_error(ytest, xgbrp, squared=False)))
```

```
RMSE Score:- 39.042231074399844
```

LightGBM

In [20]:
```
import lightgbm as lgb

lgbr = lgb.LGBMRegressor(learning_rate=0.05, n_estimators=1000, max_depth=3)
lgbr.fit(Xtrain, ytrain)

lgbp = lgbr.predict(Xtest)

print("RMSE Score:- {0}".format(mean_squared_error(ytest, lgbp, squared=False)))
```

```
RMSE Score:- 36.20811622540985
```

Hyper-Parameter Tuning:-

XGBRegressor

In [21]:
```
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV

gsc = GridSearchCV(
            estimator=XGBRegressor(),
            param_grid={"estimator__learning_rate": (0.01, 0.05), "estimator__n_estimato
            "estimator__objective": ('reg:squarederror', 'reg:gamma', 'reg:linear'), "es
            cv=3, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1)

grid_result = gsc.fit(Xtrain, ytrain)
print(grid_result.best_params_, "\n\n\n")
print("RMSE Score:- {0}".format(mean_squared_error(ytest, gsc.predict(Xtest), squared=Fa
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
[21:03:51] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.6.0/src/lea
rner.cc:627:
Parameters: { "estimator__learning_rate", "estimator__max_depth", "estimator__n_estimato
rs", "estimator__objective" } might not be used.
```

```
  This could be a false alarm, with some parameters getting used by language bindings bu
t
  then being mistakenly passed down to XGBoost core, or some parameter actually being us
ed
  but getting flagged wrongly here. Please open an issue if you find any such cases.


{'estimator__learning_rate': 0.01, 'estimator__max_depth': 5, 'estimator__n_estimators':
100, 'estimator__objective': 'reg:squarederror'}



RMSE Score:- 38.63728727448773
```

LightGBM Regressor

In [22]:

```python
from lightgbm import LGBMRegressor
from sklearn.model_selection import GridSearchCV

lgsv = GridSearchCV(
            estimator=LGBMRegressor(),
            param_grid={"estimator__boosting_type":('rf', 'dart'),"estimator__learning_r
                 "estimator__n_estimators": (50, 100), "estimator__max_bin": (32, 64),
                 "estimator__max_depth": (3, 5), "estimator__num_leaves": (32,64)},
            cv=3, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1)

lreg = lgsv.fit(Xtrain, ytrain)
print(lreg.best_params_, "\n\n\n")

print("RMSE Score:- {0}".format(mean_squared_error(ytest, lgsv.predict(Xtest), squared=F
```

```
Fitting 3 folds for each of 64 candidates, totalling 192 fits
[LightGBM] [Warning] Unknown parameter: estimator__max_depth
[LightGBM] [Warning] Unknown parameter: estimator__n_estimators
[LightGBM] [Warning] Unknown parameter: estimator__boosting_type
[LightGBM] [Warning] Unknown parameter: estimator__learning_rate
[LightGBM] [Warning] Unknown parameter: estimator__max_bin
[LightGBM] [Warning] Unknown parameter: estimator__num_leaves
{'estimator__boosting_type': 'rf', 'estimator__learning_rate': 0.05, 'estimator__max_bi
n': 32, 'estimator__max_depth': 3, 'estimator__n_estimators': 50, 'estimator__num_leave
s': 32}



RMSE Score:- 35.71700775031582
```

Feature Importances

In [23]:

```python
lgbr.feature_name_
```

Out[23]:

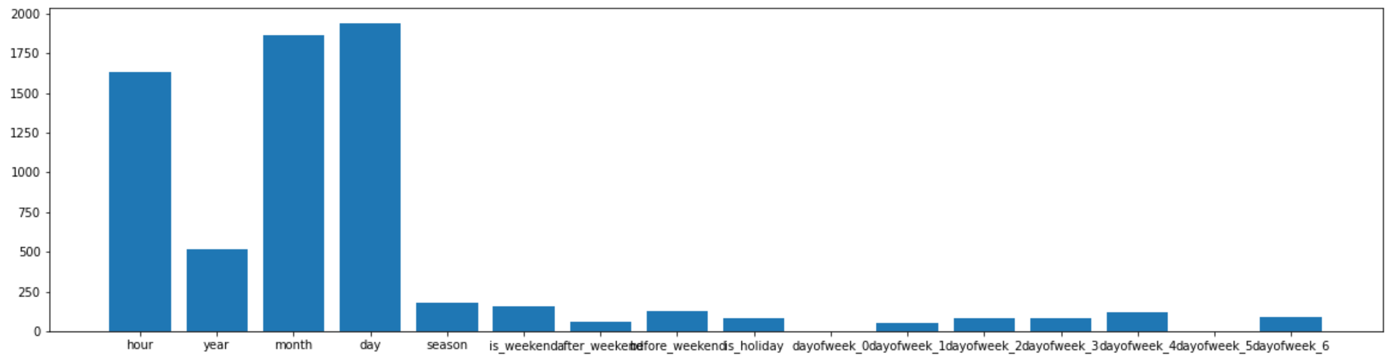
```
['hour',
 'year',
 'month',
 'day',
 'season',
 'is_weekend',
 'after_weekend',
 'before_weekend',
 'is_holiday',
 'dayofweek_0',
 'dayofweek_1',
 'dayofweek_2',
 'dayofweek_3',
 'dayofweek_4',
```

```
 'dayofweek_5',
 'dayofweek_6']
```

In [24]:
```python
import matplotlib.pyplot as plt
plt.figure(figsize=(20,5))

plt.bar([lgbr.feature_name_[x] for x in range(len(lgbr.feature_importances_))], lgbr.fea
```

Out[24]:
```
<BarContainer object of 16 artists>
```



Model Testing

Predications

Finally, I concluded that `LGBMRegressor()` is best estimator for forcasting the car rentals on hourly basis.

In [25]:
```python
testpred = lgsv.predict(test_df)
```

In [26]:
```python
test_df.shape
```

Out[26]:
```
(7650, 16)
```

Submissions

In [27]:
```python
submissions = pd.DataFrame({
    'date': test_df.index,
    'hour': test_df.hour,
    'demand': np.round(testpred, 0)
})
submissions['demand'] = submissions['demand'].astype('int')
submissions.to_csv('submission.csv', index=False)
```