**DATASET AND PREPROCESING FOR SMART PARKING**

**BATCH MEMBER:**

622621121302:R..Ragavan

**PHASE 3 DOCUMENT SUBMISSION**

**TITLE: Smart parking**

**Introduction**

I am going to analysis and visulize some datasets. In this note book, I am analyzing the time series dataset downloaded from the UCI machine learning repository.

# Time Series

A time series data is a series of data points or observations recorded at different or regular time intervals. In general, a time series is a sequence of data points taken at equally spaced time intervals. The frequency of recorded data points may be hourly, daily, weekly, monthly, quarterly or annually.

A time series analysis encompasses statistical methods for analyzing time series data. These methods enable us to extract meaningful statistics, patterns and other characteristics of the data. Time series are visualized with the help of line charts. So, time series analysis involves understanding inherent aspects of the time series data so that we can create meaningful and accurate forecasts.

# Types of Data

* A time series data means that data is recorded at different time periods or intervals. The time series data may be of three types:-
* **Time series data** - The observations of the values of a variable recorded at different points in time is called time series data.
* **Cross sectional data** - It is the data of one or more variables recorded at the same point in time.
* **Pooled data** - It is the combination of time series data and cross sectional data.

# Terminology

* There are various terms and concepts in time series that we should know. These are as follows:-
* **Dependence** - It refers to the association of two observations of the same variable at prior time periods.
* **Stationarity** - It shows the mean value of the series that remains constant over the time period. If past effects accumulate and the values increase towards infinity then stationarity is not met.
* **Differencing** - Differencing is used to make the series stationary and to control the auto-correlations. There may be some cases in time series analyses where we do not require differencing and over-differenced series can produce wrong estimates.
* **Specification** - It may involve the testing of the linear or non-linear relationships of dependent variables by using time series models such as ARIMA models.
* **Exponential Smoothing** - Exponential smoothing in time series analysis predicts the one next period value based on the past and current value. It involves averaging of data such that the non-systematic components of each individual case or observation cancel out each other. The exponential smoothing method is used to predict the short term prediction.
* **Curve fitting** - Curve fitting regression in time series analysis is used when data is in a non-linear relationship.
* **ARIMA** - ARIMA stands for Auto Regressive Integrated Moving Average.

**Attribute Information:**

SystemCodeNumber: Car park ID

Capacity: Car park capacity

Occupancy: Car park occupancy rate

LastUpdated: Date and Time of the measure

**Data Set Characteristics:** Multivariate, Univariate, Sequential, Time-Series

**Number of Instances:** 35717

**Number of Attributes:** 4

# Data Visualization

Import necessary libraries.

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

from scipy.stats import pearsonr

%matplotlib inline

Import data set.

parking\_data = pd.read\_csv('/kaggle/input/dataset.csv')

parking\_data.describe()

|  | **Capacity** | **Occupancy** |
| --- | --- | --- |
| **count** | 35717.000000 | 35717.000000 |
| **mean** | 1397.550130 | 642.228911 |
| **std** | 1179.326833 | 656.955535 |
| **min** | 220.000000 | -8.000000 |
| **25%** | 500.000000 | 210.000000 |
| **50%** | 849.000000 | 446.000000 |
| **75%** | 2009.000000 | 798.000000 |
| **max** | 4675.000000 | 4327.000000 |

Removing inconsistence data. Such as duplicate values, negative occupancy, occupancy value greater than capacity.

print('Before removing inconsistence data:',parking\_data.shape)

parking\_data.dropna(inplace = True)

parking\_data.drop\_duplicates(keep='first',inplace=True)

parking\_data = parking\_data[parking\_data['Occupancy']>=0 ]

parking\_data = parking\_data[parking\_data['Capacity']>=0 ]

false\_data = parking\_data[parking\_data['Occupancy']> parking\_data['Capacity']]

parking\_data = pd.concat([parking\_data, false\_data]).drop\_duplicates(keep=False)

print('After removing inconsistence data:',parking\_data.shape)

parking\_data.describe()

Before removing inconsistence data: (35717, 4)

After removing inconsistence data: (35116, 4)

|  | **Capacity** | **Occupancy** |
| --- | --- | --- |
| **count** | 35116.000000 | 35116.000000 |
| **mean** | 1407.923966 | 645.350496 |
| **std** | 1183.011879 | 660.531340 |
| **min** | 220.000000 | 0.000000 |
| **25%** | 577.000000 | 210.000000 |
| **50%** | 863.000000 | 450.000000 |
| **75%** | 2009.000000 | 799.000000 |
| **max** | 4675.000000 | 4327.000000 |

Convert LastUpdated column into Time and Date column. Add new columns for occupancy rate in percentage and Day of Week.

parking\_data['OccupancyRate'] = (100.0\*parking\_data['Occupancy'])/parking\_data['Capacity']

dateTime = parking\_data['LastUpdated'].str.split(" ", n = 1, expand = True)

date = dateTime[0]

time = dateTime[1]

parking\_data['Date'] = date

parking\_data['Time'] = time

day\_name = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

parking\_data['DayOfWeek'] = pd.to\_datetime(parking\_data['Date']).dt.dayofweek.apply(lambda x: day\_name[x])

parking\_data.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 35116 entries, 0 to 35716

Data columns (total 8 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 SystemCodeNumber 35116 non-null object

1 Capacity 35116 non-null int64

2 Occupancy 35116 non-null int64

3 LastUpdated 35116 non-null object

4 OccupancyRate 35116 non-null float64

5 Date 35116 non-null object

6 Time 35116 non-null object

7 DayOfWeek 35116 non-null object

dtypes: float64(1), int64(2), object(5)

memory usage: 2.4+ MB

Plot occupancy rate for all car parks with respect to time.

plt.plot([],[])

park\_name = parking\_data['SystemCodeNumber'].unique()

#print(park\_name)

for i in range(len(park\_name)):

s = park\_name[i]

rate = parking\_data[parking\_data['SystemCodeNumber'] == s]['OccupancyRate']

time=pd.to\_datetime(parking\_data[parking\_data['SystemCodeNumber'] == s]['Time'],format='%H:%M:%S')

plt.scatter(time,rate,label=s)

plt.gcf().autofmt\_xdate()

myFmt = mdates.DateFormatter('%H:%M')

plt.gca().xaxis.set\_major\_formatter(myFmt)

plt.xlabel('Time')

plt.ylabel('Occupancy Rate')

plt.show()

plt.close()

Plot mean occupancy rate with respect to each carpark.

xData = parking\_data.groupby('SystemCodeNumber')['OccupancyRate'].mean()

key\_list = list(xData.keys())

val\_list = []

for x in key\_list:

val\_list.append(xData[x])

df = pd.DataFrame(list(zip(key\_list, val\_list)),

columns =['Park ID', 'Mean Occupancy Rate'])

ax = sns.barplot(y='Park ID',x='Mean Occupancy Rate',data=df,orient="h")

ax.set(ylabel="Car Park ID", xlabel = "Mean Occupancy Rate")

ax.tick\_params(axis='y', labelsize=7)

Scatter plot of occupancy rate of **BHMBCCMKT01** car park.

for i in range(1):

s = park\_name[i]

rate = parking\_data[parking\_data['SystemCodeNumber'] == s]['OccupancyRate']

time=pd.to\_datetime(parking\_data[parking\_data['SystemCodeNumber'] == s]['Time'],format='%H:%M:%S')

plt.scatter(time,rate,label=s)

plt.gcf().autofmt\_xdate()

myFmt = mdates.DateFormatter('%H:%M')

plt.gca().xaxis.set\_major\_formatter(myFmt)

plt.xlabel('Time')

plt.ylabel('Occupancy Rate')

plt.show()

plt.close()

park\_name = ['BHMEURBRD01']

for i in range(len(park\_name)):

s = park\_name[i]

rate = parking\_data[parking\_data['SystemCodeNumber'] == s]['OccupancyRate']

time=pd.to\_datetime(parking\_data[parking\_data['SystemCodeNumber'] == s]['Date'])

plt.plot(time,rate)

plt.gcf().autofmt\_xdate()

plt.xlabel('Date')

plt.ylabel('Occupancy Rate')

plt.show()

plt.close()

There are some difference in occupancy rate in weekends and workday. I am going to plot occupancy rate for Shopping car parks on a weekends and workday.

plt.plot([],[])

rate = parking\_data[parking\_data['SystemCodeNumber'] == 'Shopping']

rate = rate[rate['Date'] == '2016-10-06']['OccupancyRate']

time = parking\_data[parking\_data['SystemCodeNumber'] == 'Shopping']

time=pd.to\_datetime(time[time['Date'] == '2016-10-06']['Time'],format='%H:%M:%S')

plt.plot(time,rate,label='2016-10-06')

rate = parking\_data[parking\_data['SystemCodeNumber'] == 'Shopping']

rate = rate[rate['Date'] == '2016-10-09']['OccupancyRate']

time = parking\_data[parking\_data['SystemCodeNumber'] == 'Shopping']

time=pd.to\_datetime(time[time['Date'] == '2016-10-09']['Time'],format='%H:%M:%S')

plt.plot(time,rate,label='2016-10-09')

plt.gcf().autofmt\_xdate()

myFmt = mdates.DateFormatter('%H:%M')

plt.gca().xaxis.set\_major\_formatter(myFmt)

plt.xlabel('Time')

plt.ylabel('Occupancy Rate')

plt.legend()

plt.show()

plt.close()

We can plot data count with respect to days of week. We can see that we get less data on weekends.

ax = sns.catplot(x='DayOfWeek',kind='count',data=parking\_data,orient="h")

ax.fig.autofmt\_xdate()

ax.set(xlabel="Week Days", ylabel = "Count")

<seaborn.axisgrid.FacetGrid at 0x7fe440656b90>

It can be clear from this box graph.

ax = sns.catplot(x = "DayOfWeek",y="OccupancyRate",kind='box',data=parking\_data)

ax.set(xlabel="Week Days", ylabel = "Occupancy Rate")

<seaborn.axisgrid.FacetGrid at 0x7fe44055d310>

We can plot heatmap for occupancy rate with respect to data for each carpark.

heatmap\_data = pd.pivot\_table(parking\_data, values='OccupancyRate',

index=['SystemCodeNumber'],

columns='Date')

ax = sns.heatmap(heatmap\_data , cmap="BuGn")

ax.set(ylabel="Car Park ID", xlabel = "Date")

[Text(32.99999999999999, 0.5, 'Car Park ID'), Text(0.5, 15.0, 'Date')]

# Data Set Preparation

test\_data = parking\_data[(pd.to\_datetime(parking\_data['Date']) >= pd.to\_datetime('2016-12-13'))]

train\_data = pd.concat([parking\_data, test\_data]).drop\_duplicates(keep=False)

print('Train data size:',train\_data.shape)

print('Test data size:',test\_data.shape)

train\_data.to\_csv('train.csv',index=False)

test\_data.to\_csv('test.csv',index=False)

Train data size: (31764, 8)

Test data size: (3352, 8)