DATA VISUALISATION for LSTM Model : dublin bikes and dublin weather

Initial Statements, Setup & File access

Words "Column", "Features", "Feature vectors" are used synonymously to indicate a feature of data.

In [1]:
```
# Import pandas, numpy, matplotlib, seaborn libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.backends.backend_pdf import PdfPages
from tabulate import tabulate
import datetime
import tensorflow as tf

# hide ipykernel warnings
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

In [2]:
```
# read in data from csv file to pandas dataframe.
bikeStatic = pd.read_csv('dBikeS.csv',  keep_default_na=True, sep=',\s+', deli
miter=',', skipinitialspace=True)
bikeDynamic = pd.read_csv('dBikeD.csv',  keep_default_na=True, sep=',\s+', del
imiter=',', skipinitialspace=True)
weather = pd.read_csv('dWeatherD.csv',  keep_default_na=True, sep=',\s+', deli
miter=',', skipinitialspace=True)
```

1. Data Quality Report

- **Scope of stage 1**
  - Data is not dropped unless for rows are duplicated.
  - Data within a feature is manipulated only with mathematical operation. Data is not altered with reference to data in other features.
  - Null values are not treated/ replaced with a unique name. They are preserved for stage2 operations.

1.1 Data view and formatting for Dublin bike dynamic data

1.1.1 Check details about number of data samples and attributes in data

In [3]:
```
bikeDynamic.shape
```

Out[3]:
```
(369160, 8)
```

1.1.2 List sample head and tail rows of data

Sample first 5 rows

In [4]:
```
bikeDynamic.head()
```

Out[4]:

	id_Entry	number	status	bike_stands	available_bike_stands	available_bikes	last_update	da
0	1745	42	OPEN	30	17	13	2020-02-28 14:55:41	
1	1746	30	OPEN	20	16	4	2020-02-28 14:59:32	
2	1747	54	OPEN	33	22	9	2020-02-28 14:54:01	
3	1748	108	OPEN	40	37	3	2020-02-28 14:59:44	
4	1749	56	OPEN	40	13	27	2020-02-28 14:52:20	

Sample last 5 rows

In [5]:
```
bikeDynamic.tail()
```

Out[5]:

	id_Entry	number	status	bike_stands	available_bike_stands	available_bikes	last_upda
369155	370900	39	OPEN	20	6	14	2020-03-2 11:37:4
369156	370901	83	OPEN	40	27	13	2020-03-2 11:35:5
369157	370902	92	OPEN	40	25	15	2020-03-2 11:38:0
369158	370903	21	OPEN	30	22	8	2020-03-2 11:31:4
369159	370904	88	OPEN	30	9	21	2020-03-2 11:35:0

Results:

- Column "id_Entry" is possibly a key column uniquely identifying a station.
- No duplicate column pairs are present in lay man observation
- Spreadsheet program shows that all values logged in database are normal and nothing unregulated found.

Observation on spreadsheet state that results for date 28 February and 24 March are patial. Hence, they are to be dropped for data consistency.

In [6]:
```
#DATETIME DATA

# Select columns containing datetime data
continous_date_columns = bikeDynamic[['last_update', 'data_entry_timestamp']].
columns

# Assign object type datetime to columns enlisted in continous_date_columns
for column in continous_date_columns:
    bikeDynamic[column] = pd.to_datetime(bikeDynamic[column])

d1 = datetime.datetime.strptime('2020-03-24 11:37:49','%Y-%m-%d %H:%M:%S')
d2 = datetime.datetime.strptime('2020-02-28 11:37:49','%Y-%m-%d %H:%M:%S')

bikeDynamic = bikeDynamic[(bikeDynamic['data_entry_timestamp'].dt.date != d1.d
ate()) & (bikeDynamic['data_entry_timestamp'].dt.date != d2.date())]
```

1.1.3 Convert features to apropriate data types

1.1.3.1 Count number of distinct values assumed by data for each feature

In [7]:
```
# Gather information related to identifiers for instacnes, count of instances
 and count of unique instances for all features.
# This information is stored into a csv.

bikeDynamic_count = pd.DataFrame(
    [column, str(bikeDynamic[column].count()), str(len(bikeDynamic[column].uni
que())),\
    round((len(bikeDynamic[column].unique()) / bikeDynamic[column].count()),6
)] for column in bikeDynamic.columns.values\
    )
bikeDynamic_count.columns = ['Features', 'Instances', 'Unique Instances','uniq
ue instances : Total instances']

bikeDynamic_count
```

Out[7]:

	Features	Instances	Unique Instances	unique instances : Total instances
0	id_Entry	355410	355410	1.000000
1	number	355410	110	0.000310
2	status	355410	2	0.000006
3	bike_stands	355410	17	0.000048
4	available_bike_stands	355410	41	0.000115
5	available_bikes	355410	41	0.000115
6	last_update	355410	300353	0.845089
7	data_entry_timestamp	355410	3231	0.009091

1.1.3.2 Enlist preassigned data types

In [8]:
```
print(tabulate(pd.DataFrame(bikeDynamic.dtypes), headers=["Feature", "Data Typ
e"]), "\n\n\n")
```

```
Feature                Data Type
---------------------  --------------
id_Entry               int64
number                 int64
status                 object
bike_stands            int64
available_bike_stands  int64
available_bikes        int64
last_update            datetime64[ns]
data_entry_timestamp   datetime64[ns]
```

1.1.3.3 Decide data types to be assigned to each feature

- id_Entry is primary keey for the dataset.
- Examination of CSV as a spreadsheet helps to **substantiate speculation** about actual data types:

Features	Data Classification	Subtype	Discription
id_Entry	numeric	discrete	Primary key for database
number	numeric	discrete	staion id
status	catagorical	nominal	station is operational or closed
bike_stands	numeric	discrete	total number of stands at station
available_bike_stands	numeric	discrete	available bikes at station
available_bikes	numeric	discrete	available parking slots at station
last_update	datetime	discrete	last update to API serivce server by station
data_entry_timestamp	datetime	discrete	time of data entry into server; not relevent for analysis

1.1.3.4 Convert to decided data type

In [9]:
```
#CATAGORICAL DATA

# Select columns containing categorical data
categorical_columns = bikeDynamic[['status']].columns

# Assign data type category to columns listed in categorical_columns
for column in categorical_columns:
    bikeDynamic[column] = bikeDynamic[column].astype('category')
```

In [10]:
```
#CONTINUOUS DATA

# Select columns containing continuous data
continous_columns = bikeDynamic[['id_Entry', 'number','bike_stands','available
_bike_stands','available_bikes']].columns

# Assign data type int64 to columns listed in continuous_columns
for column in continous_columns:
    bikeDynamic[column] = bikeDynamic[column].astype('int64')
```

1.1.3.5 Varify correct data type casting of features

In [11]:
```
print(tabulate(pd.DataFrame(bikeDynamic.dtypes), headers=["Feature", "Data Typ
e"]), "\n\n\n")
```

```
Feature                 Data Type
---------------------   --------------
id_Entry                int64
number                  int64
status                  category
bike_stands             int64
available_bike_stands   int64
available_bikes         int64
last_update             datetime64[ns]
data_entry_timestamp    datetime64[ns]
```

1.1.4 Drop duplicates

In [12]:
```
# Check for duplicate rows
#Print the number of duplicate rows, without the original rows that were dupli
cated

# Check for duplicate rows for primary key "id_Entry"
print('Number of duplicate (excluding first) rows in the table is: ', bikeDyna
mic.duplicated(subset = "id_Entry").sum())

# Use "keep=False" to mark all duplicates as true, including the original rows
that were duplicated.
print('Number of duplicate rows (including first) in the table is:', bikeDynam
ic[bikeDynamic.duplicated(subset = "id_Entry",keep=False)].shape[0])
```

```
Number of duplicate (excluding first) rows in the table is:  0
Number of duplicate rows (including first) in the table is: 0
```

In [13]:
```
# Check for duplicate columns
#First transpose the df so columns become rows, then apply the same check as a
bove
# Since cardinality of data is huge and ever increasing; and we just need to s
ee if NO DUPLICATES EXIST; hence subset of database is taken.

bikeDynamicT = bikeDynamic.head(1000).T

# Check for duplicate columns.
print("Number of duplicate (excluding first) columns in the table is: ", bikeD
ynamicT.duplicated().sum())

#Print the number of duplicates, including the original columns that were dupl
icated
print("Number of duplicate (including first) columns in the table is: ",  bike
DynamicT[bikeDynamicT.duplicated(keep=False)].shape[0])
```

```
Number of duplicate (excluding first) columns in the table is:  0
Number of duplicate (including first) columns in the table is:  0
```

Result : Duplicate columns (features) do exist

- Rows : Duplicate samples do not exist. id_Entry has (unique values : total values ratio) = 1. Logically, its a primary key for the dataset. Hence, duplicacies are checked with its respect and none are found.
- Columns : Duplicate samples do not exist.

1.1.5 Check constant features

1.1.5.1 Catagorical features

In [14]:
```
# Print table with categorical statistics
bikeDynamic.select_dtypes(['category']).describe().T
```

Out[14]:

	count	unique	top	freq
status	355410	2	OPEN	355066

Categorical Data

- Reviewing the categorical data below we can see all unique values > 1

1.1.5.2 Continuous features

```
In [15]: # Print table with continuous statistics
bikeDynamic.select_dtypes(include=['int64']).describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	
id_Entry	355410.0	185389.500000	102598.173924	7685.0	96537.25	185389.5	2742
number	355410.0	60.518182	33.767631	2.0	31.00	61.5	9
bike_stands	355410.0	32.181818	7.650539	16.0	29.00	30.0	4
available_bike_stands	355410.0	20.249658	10.792020	0.0	12.00	20.0	2
available_bikes	355410.0	11.842146	9.583294	0.0	4.00	10.0	1

Continuous Data

- No continuous feature has a non zero standard deviation.
- This implies that feature does not contain a single constant value in all of the rows. Thus in this case, none of the continuous features are constant.
- Result - No constant columns

1.1.5.3 DateTime features

```
In [16]: # Print table with continuous statistics
bikeDynamic.select_dtypes(include=['datetime']).describe().T
```

Out[16]:

	count	unique	top	freq	first	last
last_update	355410	300353	2020-03-13 21:14:38	116	2020-02-28 22:57:37	2020-03-23 23:49:05
data_entry_timestamp	355410	3231	2020-03-19 03:40:02	110	2020-02-29 00:00:06	2020-03-23 23:50:02

DateTime Data

- Reviewing the datetime data below we can see all unique values > 1

Though this is not catagorical data, it is valid to say that last_update being same for multiple stations is a likely possibility. So is case for data_entry_timestamp; which represents time of data entry into database by data scraper.

Result : No constant Features found in dataset

1.1.6 Check for null values in features

```
In [17]: print("Features".ljust(20," "),"Null instances","\n\n")
bikeDynamic.isnull().sum()
```

```
Features             Null instances
```

```
Out[17]: id_Entry                0
number                  0
status                  0
bike_stands             0
available_bike_stands   0
available_bikes         0
last_update             0
data_entry_timestamp    0
dtype: int64
```

Result : No null values found

1.2 Data cleansing and discriptive statistics

1.2.1 Varify cardinality

1.2.1.1 Catagorical features

In [18]:
```
# Check for irregular cardinality & permitted values in categorical features.
columns = list(bikeDynamic.select_dtypes(['category']).columns.values)
for column in columns:
    print("Feature:",column,"\tCardinality:",str(len(bikeDynamic[column].uniqu
e())),"\n",pd.unique(bikeDynamic[column].ravel()),"\n\n")
```

```
Feature: status 	Cardinality: 2 
 [OPEN, CLOSED]
Categories (2, object): [OPEN, CLOSED]
```

In [19]:
```
# For each catagorical feature, display the number of instances each of its va
lues has.
columns = list(bikeDynamic.select_dtypes(['category']).columns.values)
for column in columns:
    featureDetail = column+"   Cardinality:"+str(len(bikeDynamic[column].uniqu
e()))
    print(featureDetail,"\n{}\n".format('-'*len(str(featureDetail))))
    print(tabulate(pd.DataFrame(bikeDynamic[column].value_counts().nlargest(15
)), headers=["Instance", "Number of Instances"]), "\n\n\n")
```

```
status   Cardinality:2 
---------------------- 

Instance      Number of Instances
----------  ---------------------
OPEN                       355066
CLOSED                        344
```

Values of cardinality of catagorical features are regular and within normal consideration.

- Almost stations are 'OPEN'.
- Only 0.1% times station entry is 'CLOSED'

1.2.1.2 Continuous features

In [20]:
```
# Check for irregular cardinality & permitted values in continuous features.
columns = list(bikeDynamic.select_dtypes(['int64']).columns.values)
for column in columns:
    print("Feature:",column,"\tCardinality:",str(len(bikeDynamic[column].unique())),"\n",pd.unique(sorted(bikeDynamic[column].ravel())),"\n\n\n")
```

```
Feature: id_Entry       Cardinality: 355410
 [  7685   7686   7687 ... 363092 363093 363094]



Feature: number         Cardinality: 110
 [  2   3   4   5   6   7   8   9  10  11  12  13  15  16  17  18  19  21
  22  23  24  25  26  27  28  29  30  31  32  33  34  36  37  38  39  40
  41  42  43  44  45  47  48  49  50  51  52  53  54  55  56  57  58  59
  61  62  63  64  65  66  67  68  69  71  72  73  74  75  76  77  78  79
  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97
  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115
 116 117]



Feature: bike_stands    Cardinality: 17
 [16 20 21 22 23 24 25 27 29 30 31 32 33 35 36 38 40]



Feature: available_bike_stands  Cardinality: 41
 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40]



Feature: available_bikes        Cardinality: 41
 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40]
```

In [21]:
```
# For each continuous feature, display the number of instances each of its val
ues has.
columns = list(bikeDynamic.select_dtypes(['int64']).columns.values)
for column in columns:
    print(column,"\n{}\n".format('-'*len(str(column))))
    print(tabulate(pd.DataFrame(bikeDynamic[column].value_counts().nlargest(15
)), headers=["Feature", "Number of Instances"]), "\n\n\n")
```

id_Entry

Feature	Number of Instances
8188	1
287449	1
271057	1
277202	1
275155	1
264916	1
262869	1
269014	1
266967	1
289496	1
293594	1
316111	1
291547	1
281308	1
279261	1

number

Feature	Number of Instances
117	3231
45	3231
33	3231
34	3231
36	3231
37	3231
38	3231
39	3231
40	3231
41	3231
42	3231
43	3231
44	3231
47	3231
31	3231

bike_stands

Feature	Number of Instances
40	138933
30	87237
20	51696
29	12924
38	9693
35	6462
25	6462
23	6462
36	6462
16	6462
33	3231
32	3231
31	3231
27	3231
24	3231

available_bike_stands

Feature	Number of Instances

20	15713
30	13632
40	12793
19	12684
17	12199
18	12105
16	11880
29	11317
27	10815
28	10763
14	10048
15	9963
13	9763
26	9756
23	9735

available_bikes

Feature	Number of Instances
0	36620
1	17788
2	16171
3	14873
4	14343
9	14273
6	14053
7	13839
8	13592
10	13546
11	13496
5	13048
12	12804
13	11438
14	11377

Values of cardinality of Continuous features are regular and within normal consideration.

- No irregularity is found
- Refering to quartile ranges and permissible values, it is evident that for both bike_stands, available_bike_stands and available_bikes features:
  - On average, nearly 35% of stands have 0 available bikes at any time
  - 70% Bike stands have capacity greater than 29; 39% Bike stands have capcity of 40 bikes

1.2.1.3 DateTime features

In [22]:
```
# Check for irregular cardinality & permitted values in datetime features.
columns = list(bikeDynamic.select_dtypes(['datetime64']).columns.values)
for column in columns:
    print("Feature:",column,"\tCardinality:",str(len(bikeDynamic[column].unique())),"\n",pd.unique(sorted(bikeDynamic[column].ravel())),"\n\n\n")
```

```
Feature: last_update    Cardinality: 300353
 ['2020-02-28T22:57:37.000000000' '2020-02-28T23:50:00.000000000'
 '2020-02-28T23:50:12.000000000' ... '2020-03-23T23:48:59.000000000'
 '2020-03-23T23:49:02.000000000' '2020-03-23T23:49:05.000000000']



Feature: data_entry_timestamp   Cardinality: 3231
 ['2020-02-29T00:00:06.000000000' '2020-02-29T00:10:02.000000000'
 '2020-02-29T00:20:02.000000000' ... '2020-03-23T23:30:02.000000000'
 '2020-03-23T23:40:02.000000000' '2020-03-23T23:50:02.000000000']
```

In [23]:
```
# For each datetime feature, display the number of instances each of its values has.
columns = list(bikeDynamic.select_dtypes(['datetime64']).columns.values)
for column in columns:
    print(column,"\n{}\n".format('-'*len(str(column))))
    print(tabulate(pd.DataFrame(bikeDynamic[column].value_counts().nlargest(10)), headers=["Feature", "Number of Instances"]), "\n\n\n")
```

```
last_update
-----------

Feature                Number of Instances
-------------------  ---------------------
2020-03-13 21:14:38                    116
2020-03-06 19:32:44                     41
2020-03-14 04:06:25                     40
2020-03-16 03:55:32                     21
2020-03-16 19:33:46                     18
2020-03-12 15:31:12                     16
2020-03-16 03:55:04                     15
2020-03-16 03:59:52                     15
2020-03-13 06:58:33                     15
2020-03-16 03:54:54                     14



data_entry_timestamp
--------------------

Feature                Number of Instances
-------------------  ---------------------
2020-03-19 03:40:02                    110
2020-03-07 22:30:02                    110
2020-03-07 07:30:02                    110
2020-03-07 05:00:04                    110
2020-03-16 00:00:05                    110
2020-03-15 20:20:02                    110
2020-03-18 06:40:02                    110
2020-03-04 23:50:02                    110
2020-03-20 16:00:06                    110
2020-03-19 15:10:02                    110
```

Values of cardinality of DateTime features are regular and within normal consideration.

- As expected, data_entry_timestamp has fixed number of instances for each entry since data is entered for each station exactly once
- last_update feature shows 116 instances for a perticular "2020-03-13 21:14:38" which is an eye catcher.

1.2.2 Check logical integrity of data

Data integrity is checked for following cases:

- is "bike_stands" $>=$ "available_bike_stands" $+$ "available_bikes" [Any other sequence is incorrect]
- is "last_update" $<=$ "data_entry_timestamp" [Any other sequence is incorrect]

Date of birth for an animal must always be smaller than or equal to date of intake into shelter.

In [24]:
```
test_1 = bikeDynamic[["available_bike_stands","available_bikes","bike_stands"
]][bikeDynamic["available_bike_stands"].add(bikeDynamic["available_bikes"], ax
is=0)  >  bikeDynamic["bike_stands"]]
print("Number of rows failing the test: ", test_1.shape[0])
test_1.head(5)
```

```
Number of rows failing the test:  18
```

Out[24]:

	available_bike_stands	available_bikes	bike_stands
63885	17	0	16
248795	16	1	16
248905	16	1	16
249125	16	1	16
249235	16	1	16

Date of birth for an animal must always be smaller than or equal to date of Outcome from shelter.

In [25]:
```
test_2 = bikeDynamic[["last_update","data_entry_timestamp"]][bikeDynamic["data
_entry_timestamp"]  <  bikeDynamic["last_update"]]
print("Number of rows failing the test: ", test_2.shape[0])
test_2.head(5)
```

```
Number of rows failing the test:  0
```

Out[25]:

last_update	data_entry_timestamp

1.2.3 Save discriptive statistics into CSV for data quality report

1.2.3.1 Discriptive statistics for Catagorical Data

In [26]:
```
# Print table with categorical statistics and preserve in a csv
dStat_catagorical = bikeDynamic.select_dtypes(['category']).describe().T
dStat_catagorical.to_csv("categoricalFeatureDescription.csv")
dStat_catagorical
```

Out[26]:

	count	unique	top	freq
status	355410	2	OPEN	355066

1.2.3.2 Discriptive statistics for Continuous Data

In [27]:
```
# Print table with continuous statistics and preserve in a csv
dStat_continuous = bikeDynamic.select_dtypes(['int64']).describe().T
dStat_continuous.to_csv("continuousFeatureDescription.csv")
dStat_continuous
```

Out[27]:

	count	mean	std	min	25%	50%	
id_Entry	355410.0	185389.500000	102598.173924	7685.0	96537.25	185389.5	2742
number	355410.0	60.518182	33.767631	2.0	31.00	61.5	9
bike_stands	355410.0	32.181818	7.650539	16.0	29.00	30.0	4
available_bike_stands	355410.0	20.249658	10.792020	0.0	12.00	20.0	2
available_bikes	355410.0	11.842146	9.583294	0.0	4.00	10.0	1

1.2.3.3 Discriptive statistics for DateTime Data

1.2.3.4 Save final CSV from stage 1

In [28]:
```
# Print table with datetime statistics and preserve in a csv
dStat_datetime = bikeDynamic.select_dtypes(['datetime64']).describe().T
dStat_datetime.to_csv("datetimeFeatureDescription.csv")
dStat_datetime
```

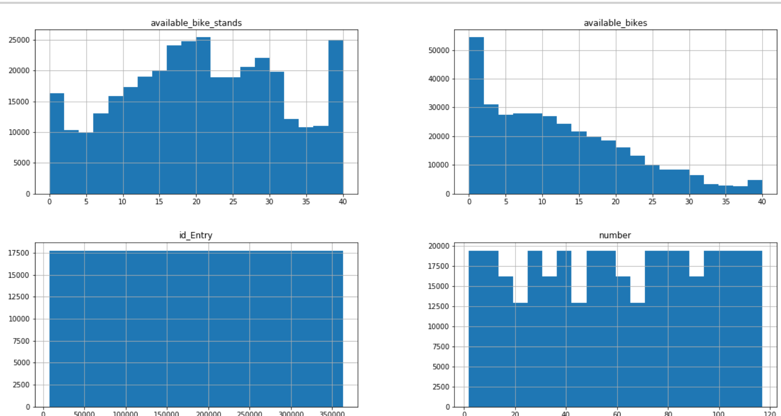
Out[28]:

	count	unique	top	freq	first	last
last_update	355410	300353	2020-03-13 21:14:38	116	2020-02-28 22:57:37	2020-03-23 23:49:05
data_entry_timestamp	355410	3231	2020-03-19 03:40:02	110	2020-02-29 00:00:06	2020-03-23 23:50:02

1.3 Graphs

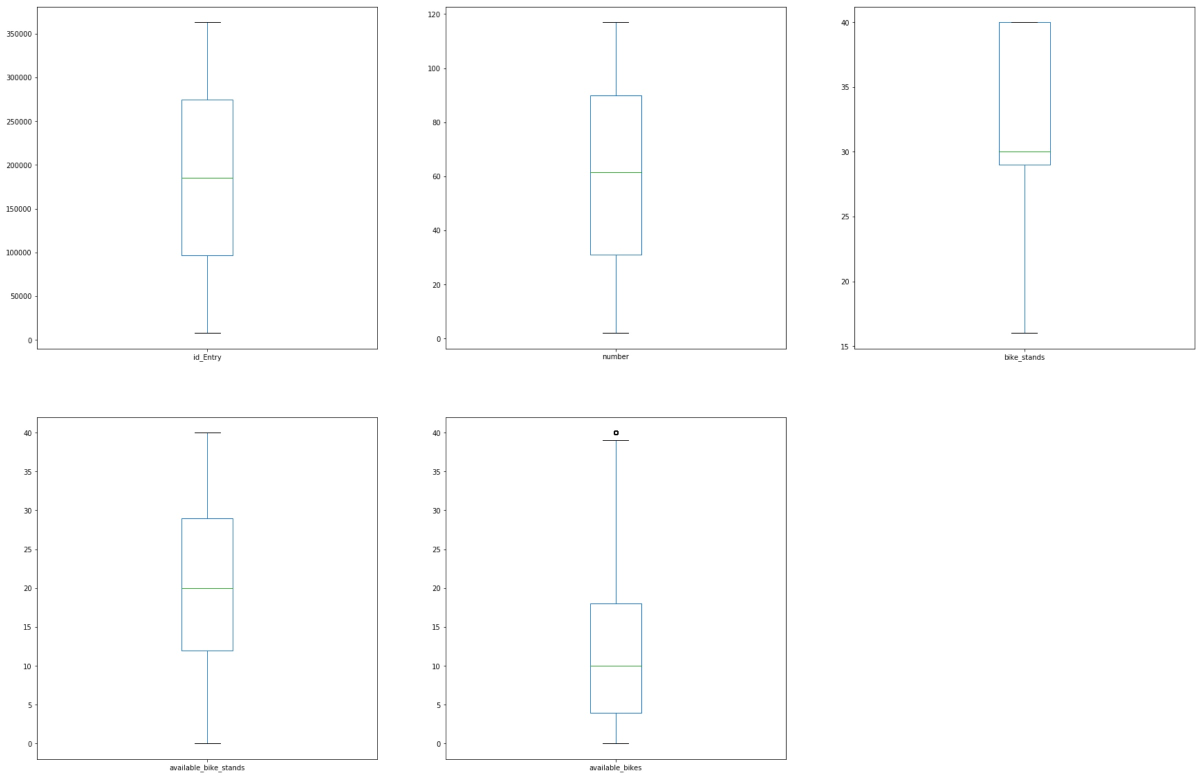
1.3.1 Save Histogram summary sheets for Continuous features into pdf file

In [29]:
```
# Plot a histogram summary sheet of the continuous features and save in a pdf
 file
columns = list(bikeDynamic.select_dtypes(['int64']).columns.values)
bikeDynamic[columns].hist(layout=(2, 3), figsize=(30,10), bins=20)
plt.savefig('continuous_histograms_1-1.pdf')
```



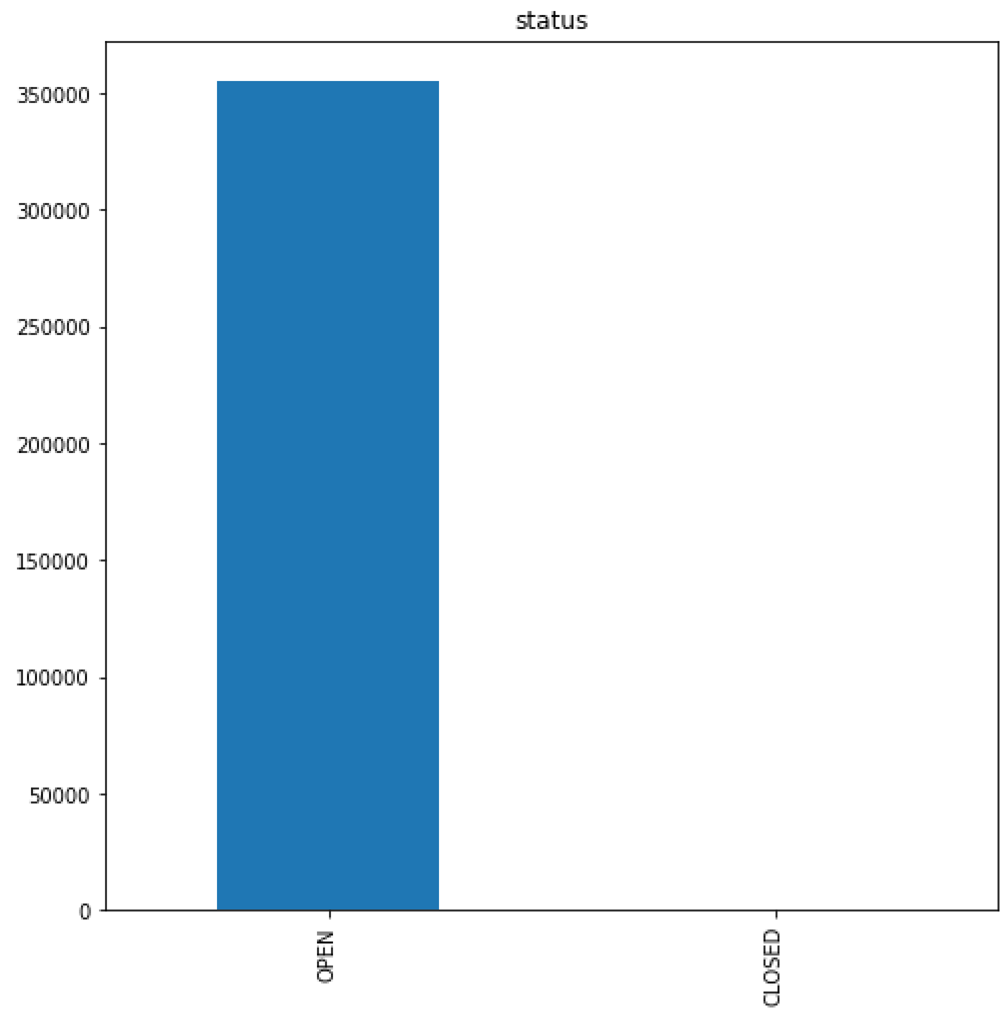
1.3.2 Save Box plot summary sheets for Continuous features into pdf file

In [30]:
```python
# Plot a histogram summary sheet of the continuous features and save in a pdf
 file
columns = list(bikeDynamic.select_dtypes(['int64']).columns.values)
bikeDynamic[columns].plot(kind='box', subplots=True, figsize=(30,20), layout=(
2,3), sharex=False, sharey=False)
plt.savefig('continuous_box_1-1.pdf')
```



1.3.5 Save Bar plot summary sheets for Catagorical features into pdf file

In [31]:
```
# Plot bar plots for all the catagorical features and save them in a single PD
F file
columns = list(bikeDynamic.select_dtypes(['category']).columns.values)
with PdfPages('categorical_bar_1-1.pdf') as pp:
    for column in columns:
        f = bikeDynamic[column].value_counts().plot(kind='bar', figsize=(8,8))
        plt.title(column)
        pp.savefig(f.get_figure())
        plt.show()
```



2. Data Quality Plan

2.1 Data findings

2.1.1 Summary of list of issues found in Data Quality Report

- test_1 states that 18 entries have bad data regarding total number of bike stands at a station.

2.1.2 Proposed solutions to rectify identified problems

- Replace feature bike_stands with value sum of available_bike_stands,available_bikes

2.2 Apply solutions to address data quality issues

2.2.1 DateTime Intake > DateTime Outcome

In [32]:
```
dTest = bikeDynamic[["id_Entry","available_bike_stands","available_bikes","bik
e_stands"]][bikeDynamic["available_bike_stands"].add(bikeDynamic["available_bi
kes"], axis=0)  >  bikeDynamic["bike_stands"]]
print("Number of rows failing the test: ", dTest.shape[0])
dTest.head(5)
```

```
Number of rows failing the test:  18
```

Out[32]:

	id_Entry	available_bike_stands	available_bikes	bike_stands
63885	65630	17	0	16
248795	250540	16	1	16
248905	250650	16	1	16
249125	250870	16	1	16
249235	250980	16	1	16

Replacing corresponding entries for feature "bike_stands":

In [33]:
```
dTest_1 = dTest["id_Entry"]

for data in dTest_1:
    bikeDynamic.loc[(bikeDynamic.id_Entry == data),'bike_stands'] = bikeDynami
c.loc[(bikeDynamic.id_Entry == data),'available_bikes'] + bikeDynamic.loc[(bik
eDynamic.id_Entry == data),'available_bike_stands']
```

In [34]:
```
dTest = bikeDynamic[["id_Entry","available_bike_stands","available_bikes","bik
e_stands"]][bikeDynamic["available_bike_stands"].add(bikeDynamic["available_bi
kes"], axis=0)  >  bikeDynamic["bike_stands"]]
print("Number of rows failing the test: ", dTest.shape[0])
dTest.head(5)
```

```
Number of rows failing the test:  0
```

Out[34]:

id_Entry	available_bike_stands	available_bikes	bike_stands

2.3 Summary of data quality plan

Feature	Data Quality issue	Solution Strategy
id_Entry	[Primary key]	Do nothing
number	Nothing	Do nothing
status	Nothing	Do nothing
bike_stands	Value inconsistancy for 18 entires	increment by 1
available_bike_stands	Nothing	Do nothing
available_bikes	Nothing	Do nothing
last_update	Nothing	Do nothing
data_entry_timestamp	Nothing	Do nothing

2.4 Save cleaned data to new CSV

In [35]:
```
bikeDynamic.to_csv('dBikeD_2.4_cleaned.csv', index=False, index_label = True)
```

3. Feature Exploration

3.1 Time series view

In [36]:
```
bikeDynamicTS = bikeDynamic
bikeDynamicTS['date_UTC'] = [datetime.datetime.timestamp(d) for d in bikeDynam
icTS['data_entry_timestamp']]
bikeDynamicTS['dayNumber'] = bikeDynamicTS['data_entry_timestamp'].dt.dayofwee
k
bikeDynamicTS.head()
```

Out[36]:

	id_Entry	number	status	bike_stands	available_bike_stands	available_bikes	last_update
5940	7685	42	OPEN	30	10	20	2020-02-28 23:52:29
5941	7686	30	OPEN	20	20	0	2020-02-28 23:53:40
5942	7687	54	OPEN	33	29	4	2020-02-28 23:54:52
5943	7688	108	OPEN	40	31	9	2020-02-28 22:57:37
5944	7689	56	OPEN	40	37	3	2020-02-28 23:50:26

In [37]:
```
bikeDynamic.to_csv('dBikeD_3.1_1_cleaned.csv', index=False, index_label = True
)
```

In [38]:
```
dates = [datetime.datetime.fromtimestamp(timestamp) for timestamp in bikeDynam
icTS['date_UTC'].unique()]
```

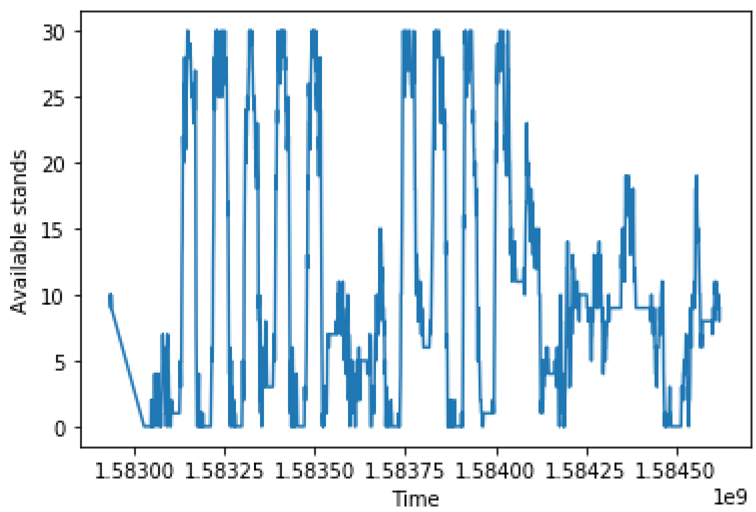
In [39]:
```
stationNumbers = bikeDynamicTS['number'].unique()
timeT = []
available_bike_stands = []

for station in stationNumbers:
    tempTime = bikeDynamicTS.loc[(bikeDynamicTS.number == station)]['date_UTC'
].values.tolist()
    tempStands = bikeDynamicTS.loc[(bikeDynamicTS.number == station)]['availab
le_bike_stands'].values.tolist()
    timeT.append(tempTime)
    available_bike_stands.append(tempStands)
```

Bikestand availability for a day

- **As seen, weekday traffic experiences a definite cycle**
- **Weekend traffic is not heavy**

Due to COVID 19 situation, "Stay at home" directive from Government of Ireland was applied in last week of March 2020. dublin bike usage was unnaturlly altered which can be seen at end of time series.

In [40]:
```
# Split data into training and testing data. Display train data
time,series = timeT[0],available_bike_stands[0]

tLen = len(time)
split_time = tLen - int(tLen*0.2)
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

plt.plot(time_train, x_train)
# plt.xticks(range(len(time_train)), time_train)
plt.xlabel('Time')
plt.ylabel('Available stands')
plt.show()

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```



In [41]:
```
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    series = tf.expand_dims(series, axis=-1)
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda w: (w[:-1], w[1:]))
    return ds.batch(batch_size).prefetch(1)
```

In [42]:
```
def model_forecast(model, series, window_size):
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size))
    ds = ds.batch(32).prefetch(1)
    forecast = model.predict(ds)
    return forecast
```

Train LSTM model

In [43]:
```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)
train_set = windowed_dataset(x_train, window_size=30, batch_size=100, shuffle_
buffer=shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=30, kernel_size=5,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.LSTM(30, return_sequences=True),
  tf.keras.layers.LSTM(30, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=150)
```

```
Epoch 1/150
26/26 [==============================] - 3s 133ms/step - loss: 10.2887 - mae:
10.8553
Epoch 2/150
26/26 [==============================] - 1s 40ms/step - loss: 3.1384 - mae:
3.6088
Epoch 3/150
26/26 [==============================] - 1s 37ms/step - loss: 2.2388 - mae:
2.6825
Epoch 4/150
26/26 [==============================] - 1s 42ms/step - loss: 1.9057 - mae:
2.3398
Epoch 5/150
26/26 [==============================] - 1s 45ms/step - loss: 1.7339 - mae:
2.1562
Epoch 6/150
26/26 [==============================] - 1s 41ms/step - loss: 1.6007 - mae:
2.0108
Epoch 7/150
26/26 [==============================] - 1s 39ms/step - loss: 1.5075 - mae:
1.9267
Epoch 8/150
26/26 [==============================] - 1s 46ms/step - loss: 1.4087 - mae:
1.8221
Epoch 9/150
26/26 [==============================] - 1s 44ms/step - loss: 1.3426 - mae:
1.7530
Epoch 10/150
26/26 [==============================] - 1s 41ms/step - loss: 1.2910 - mae:
1.7021
Epoch 11/150
26/26 [==============================] - 1s 39ms/step - loss: 1.2292 - mae:
1.6335
Epoch 12/150
26/26 [==============================] - 1s 40ms/step - loss: 1.1836 - mae:
1.5842
Epoch 13/150
26/26 [==============================] - 1s 41ms/step - loss: 1.1405 - mae:
1.5436
Epoch 14/150
26/26 [==============================] - 1s 43ms/step - loss: 1.1067 - mae:
1.5031
Epoch 15/150
26/26 [==============================] - 1s 40ms/step - loss: 1.0825 - mae:
1.4769
Epoch 16/150
26/26 [==============================] - 1s 40ms/step - loss: 1.0852 - mae:
1.4863
Epoch 17/150
26/26 [==============================] - 1s 39ms/step - loss: 1.0385 - mae:
1.4326
Epoch 18/150
26/26 [==============================] - 1s 39ms/step - loss: 1.0205 - mae:
1.4095
Epoch 19/150
26/26 [==============================] - 1s 41ms/step - loss: 1.0076 - mae:
1.3984
Epoch 20/150
26/26 [==============================] - 1s 41ms/step - loss: 1.0035 - mae:
1.3955
Epoch 21/150
26/26 [==============================] - 1s 39ms/step - loss: 0.9787 - mae:
1.3678
Epoch 22/150
26/26 [==============================] - 1s 39ms/step - loss: 0.9853 - mae:
1.3761
Epoch 23/150
26/26 [==============================] - 1s 39ms/step - loss: 0.9614 - mae:
1.3509
Epoch 24/150
26/26 [==============================] - 1s 45ms/step - loss: 0.9562 - mae:
1.3435
Epoch 25/150
26/26 [==============================] - 1s 39ms/step - loss: 0.9544 - mae:
```

```
1.3432
Epoch 26/150
26/26 [==============================] - 1s 39ms/step - loss: 0.9576 - mae:
1.3493
Epoch 27/150
26/26 [==============================] - 1s 39ms/step - loss: 1.1278 - mae:
1.5468
Epoch 28/150
26/26 [==============================] - 1s 39ms/step - loss: 0.9359 - mae:
1.3221
Epoch 29/150
26/26 [==============================] - 1s 44ms/step - loss: 0.9082 - mae:
1.2906
Epoch 30/150
26/26 [==============================] - 1s 41ms/step - loss: 0.8976 - mae:
1.2768
Epoch 31/150
26/26 [==============================] - 1s 41ms/step - loss: 0.9024 - mae:
1.2873
Epoch 32/150
26/26 [==============================] - 1s 41ms/step - loss: 0.9163 - mae:
1.3059
Epoch 33/150
26/26 [==============================] - 1s 43ms/step - loss: 0.8884 - mae:
1.2689
Epoch 34/150
26/26 [==============================] - 1s 43ms/step - loss: 0.8745 - mae:
1.2524
Epoch 35/150
26/26 [==============================] - 1s 39ms/step - loss: 0.8699 - mae:
1.2462
Epoch 36/150
26/26 [==============================] - 1s 44ms/step - loss: 0.8750 - mae:
1.2580
Epoch 37/150
26/26 [==============================] - 1s 43ms/step - loss: 0.8732 - mae:
1.2548
Epoch 38/150
26/26 [==============================] - 1s 45ms/step - loss: 0.8535 - mae:
1.2323
Epoch 39/150
26/26 [==============================] - 1s 44ms/step - loss: 0.8421 - mae:
1.2199
Epoch 40/150
26/26 [==============================] - 1s 39ms/step - loss: 0.8328 - mae:
1.2047
Epoch 41/150
26/26 [==============================] - 1s 39ms/step - loss: 0.8608 - mae:
1.2431
Epoch 42/150
26/26 [==============================] - 1s 39ms/step - loss: 0.8491 - mae:
1.2302
Epoch 43/150
26/26 [==============================] - 1s 42ms/step - loss: 0.8202 - mae:
1.1934
Epoch 44/150
26/26 [==============================] - 1s 41ms/step - loss: 0.8340 - mae:
1.2115
Epoch 45/150
26/26 [==============================] - 1s 38ms/step - loss: 0.8175 - mae:
1.1887
Epoch 46/150
26/26 [==============================] - 1s 39ms/step - loss: 0.8351 - mae:
1.2117
Epoch 47/150
26/26 [==============================] - 1s 39ms/step - loss: 0.8000 - mae:
1.1699
Epoch 48/150
26/26 [==============================] - 1s 39ms/step - loss: 0.8023 - mae:
1.1720
Epoch 49/150
26/26 [==============================] - 1s 41ms/step - loss: 0.8089 - mae:
1.1834
Epoch 50/150
```

```
26/26 [==============================] - 1s 39ms/step - loss: 0.7948 - mae:
1.1675
Epoch 51/150
26/26 [==============================] - 1s 38ms/step - loss: 0.7885 - mae:
1.1586
Epoch 52/150
26/26 [==============================] - 1s 38ms/step - loss: 0.7916 - mae:
1.1620
Epoch 53/150
26/26 [==============================] - 1s 41ms/step - loss: 0.8056 - mae:
1.1824
Epoch 54/150
26/26 [==============================] - 1s 41ms/step - loss: 0.7891 - mae:
1.1631
Epoch 55/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7674 - mae:
1.1339
Epoch 56/150
26/26 [==============================] - 1s 39ms/step - loss: 0.7677 - mae:
1.1316
Epoch 57/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7637 - mae:
1.1284
Epoch 58/150
26/26 [==============================] - 1s 42ms/step - loss: 0.7658 - mae:
1.1349
Epoch 59/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7671 - mae:
1.1397
Epoch 60/150
26/26 [==============================] - 1s 41ms/step - loss: 0.7649 - mae:
1.1369
Epoch 61/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7541 - mae:
1.1199
Epoch 62/150
26/26 [==============================] - 1s 41ms/step - loss: 0.7539 - mae:
1.1200
Epoch 63/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7481 - mae:
1.1152
Epoch 64/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7436 - mae:
1.1102
Epoch 65/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7401 - mae:
1.1019
Epoch 66/150
26/26 [==============================] - 1s 38ms/step - loss: 0.7349 - mae:
1.1004
Epoch 67/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7309 - mae:
1.0927
Epoch 68/150
26/26 [==============================] - 1s 39ms/step - loss: 0.7273 - mae:
1.0882
Epoch 69/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7698 - mae:
1.1469
Epoch 70/150
26/26 [==============================] - 1s 41ms/step - loss: 0.7272 - mae:
1.0880
Epoch 71/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7222 - mae:
1.0826
Epoch 72/150
26/26 [==============================] - 1s 42ms/step - loss: 0.7232 - mae:
1.0893
Epoch 73/150
26/26 [==============================] - 1s 39ms/step - loss: 0.7144 - mae:
1.0745
Epoch 74/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7161 - mae:
1.0771
```

```
Epoch 75/150
26/26 [==============================] - 1s 41ms/step - loss: 0.7097 - mae:
1.0698
Epoch 76/150
26/26 [==============================] - 1s 39ms/step - loss: 0.7158 - mae:
1.0795
Epoch 77/150
26/26 [==============================] - 1s 39ms/step - loss: 0.7249 - mae:
1.0937
Epoch 78/150
26/26 [==============================] - 1s 39ms/step - loss: 0.7173 - mae:
1.0843
Epoch 79/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7011 - mae:
1.0612
Epoch 80/150
26/26 [==============================] - 1s 43ms/step - loss: 0.7095 - mae:
1.0730
Epoch 81/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6989 - mae:
1.0559
Epoch 82/150
26/26 [==============================] - 1s 42ms/step - loss: 0.6912 - mae:
1.0485
Epoch 83/150
26/26 [==============================] - 1s 40ms/step - loss: 0.7007 - mae:
1.0614
Epoch 84/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6892 - mae:
1.0462
Epoch 85/150
26/26 [==============================] - 1s 42ms/step - loss: 0.6957 - mae:
1.0561
Epoch 86/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6814 - mae:
1.0359
Epoch 87/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6831 - mae:
1.0385
Epoch 88/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6896 - mae:
1.0509
Epoch 89/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6856 - mae:
1.0452
Epoch 90/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6825 - mae:
1.0396
Epoch 91/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6802 - mae:
1.0358
Epoch 92/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6911 - mae:
1.0511
Epoch 93/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6994 - mae:
1.0628
Epoch 94/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6678 - mae:
1.0208
Epoch 95/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6652 - mae:
1.0162
Epoch 96/150
26/26 [==============================] - 1s 41ms/step - loss: 0.6683 - mae:
1.0213
Epoch 97/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6652 - mae:
1.0169
Epoch 98/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6794 - mae:
1.0394
Epoch 99/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6780 - mae:
```

```
1.0357
Epoch 100/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6625 - mae:
1.0144
Epoch 101/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6906 - mae:
1.0572
Epoch 102/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6775 - mae:
1.0380
Epoch 103/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6636 - mae:
1.0181
Epoch 104/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6538 - mae:
1.0080
Epoch 105/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6577 - mae:
1.0072
Epoch 106/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6729 - mae:
1.0326
Epoch 107/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6689 - mae:
1.0279
Epoch 108/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6546 - mae:
1.0072
Epoch 109/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6521 - mae:
1.0004
Epoch 110/150
26/26 [==============================] - 1s 41ms/step - loss: 0.6578 - mae:
1.0151
Epoch 111/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6549 - mae:
1.0100
Epoch 112/150
26/26 [==============================] - 1s 41ms/step - loss: 0.6447 - mae:
0.9970
Epoch 113/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6554 - mae:
1.0117
Epoch 114/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6857 - mae:
1.0569
Epoch 115/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6486 - mae:
0.9998
Epoch 116/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6444 - mae:
0.9939
Epoch 117/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6384 - mae:
0.9857
Epoch 118/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6575 - mae:
1.0184
Epoch 119/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6394 - mae:
0.9897
Epoch 120/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6365 - mae:
0.9847
Epoch 121/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6350 - mae:
0.9839
Epoch 122/150
26/26 [==============================] - 1s 42ms/step - loss: 0.6390 - mae:
0.9896
Epoch 123/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6395 - mae:
0.9888
Epoch 124/150
```

```
26/26 [==============================] - 1s 38ms/step - loss: 0.6403 - mae:
0.9901
Epoch 125/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6299 - mae:
0.9755
Epoch 126/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6353 - mae:
0.9851
Epoch 127/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6269 - mae:
0.9723
Epoch 128/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6319 - mae:
0.9788
Epoch 129/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6302 - mae:
0.9788
Epoch 130/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6303 - mae:
0.9818
Epoch 131/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6432 - mae:
0.9978
Epoch 132/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6264 - mae:
0.9717
Epoch 133/150
26/26 [==============================] - 1s 41ms/step - loss: 0.6217 - mae:
0.9688
Epoch 134/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6285 - mae:
0.9771: 0s - loss: 0.6538 - mae: 1.
Epoch 135/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6220 - mae:
0.9679
Epoch 136/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6236 - mae:
0.9691
Epoch 137/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6224 - mae:
0.9660
Epoch 138/150
26/26 [==============================] - 1s 46ms/step - loss: 0.6302 - mae:
0.9835
Epoch 139/150
26/26 [==============================] - 1s 42ms/step - loss: 0.6205 - mae:
0.9631
Epoch 140/150
26/26 [==============================] - 1s 46ms/step - loss: 0.6253 - mae:
0.9713
Epoch 141/150
26/26 [==============================] - 1s 41ms/step - loss: 0.6187 - mae:
0.9621
Epoch 142/150
26/26 [==============================] - 1s 56ms/step - loss: 0.6229 - mae:
0.9740: 0s - loss: 0.6341 - mae: 0.9
Epoch 143/150
26/26 [==============================] - 1s 47ms/step - loss: 0.6154 - mae:
0.9573
Epoch 144/150
26/26 [==============================] - 1s 39ms/step - loss: 0.6280 - mae:
0.9786
Epoch 145/150
26/26 [==============================] - 1s 41ms/step - loss: 0.6192 - mae:
0.9657
Epoch 146/150
26/26 [==============================] - 1s 40ms/step - loss: 0.6169 - mae:
0.9651
Epoch 147/150
26/26 [==============================] - 1s 42ms/step - loss: 0.6225 - mae:
0.9715
Epoch 148/150
26/26 [==============================] - 1s 42ms/step - loss: 0.6136 - mae:
0.9573
```

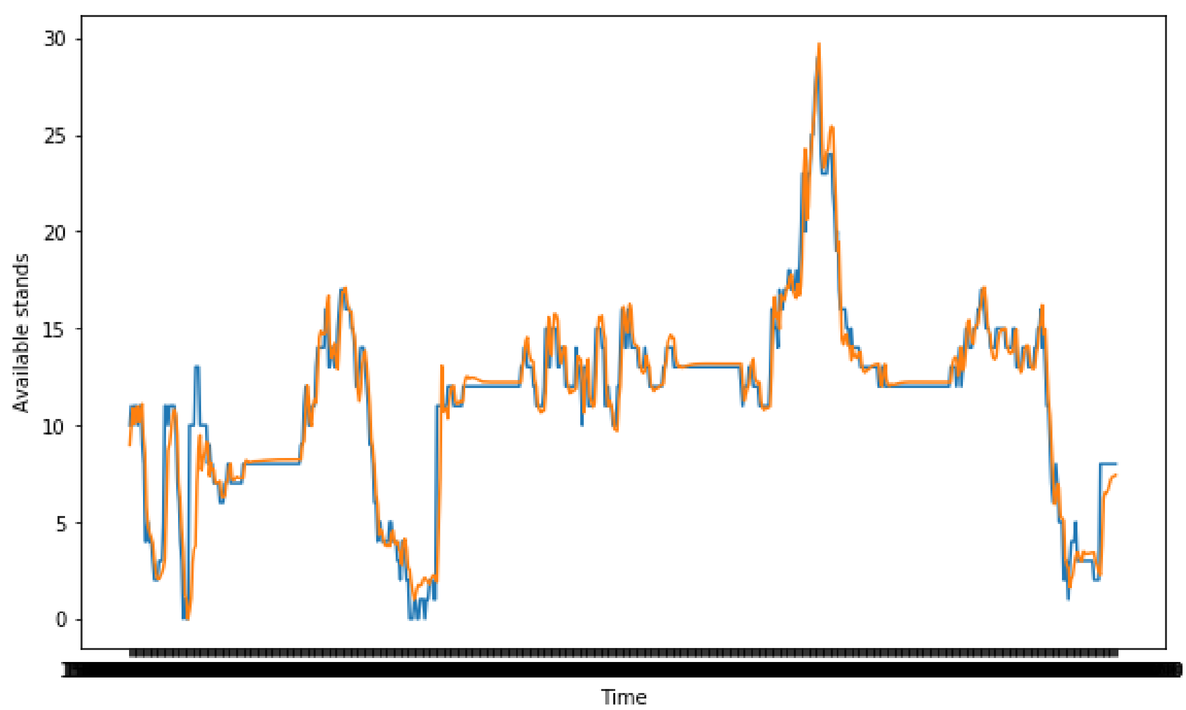
```
Epoch 149/150
26/26 [==============================] - 1s 38ms/step - loss: 0.6124 - mae:
0.9562
Epoch 150/150
26/26 [==============================] - 1s 37ms/step - loss: 0.6170 - mae:
0.9638
```

Forecast results for validation data and compare performance metrics

In [44]:
```
series = np.asarray(series)
rnn_forecast = model_forecast(model, series[..., np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_time - window_size:-1, -1, 0]
```

In [45]:
```
def plot_series(time, series, format="-", start=0, end=None):
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.grid(True)
```

In [46]:
```
plt.figure(figsize=(10, 6))

plt.plot(x_valid)
plt.xticks(range(len(time_valid)), time_valid)
plt.xlabel('Time')
plt.ylabel('Available stands')

plt.plot(rnn_forecast)
plt.xticks(range(len(time_valid)), time_valid)
plt.xlabel('Time')
plt.ylabel('Available stands')

plt.show()
```



Prediction performance metrics

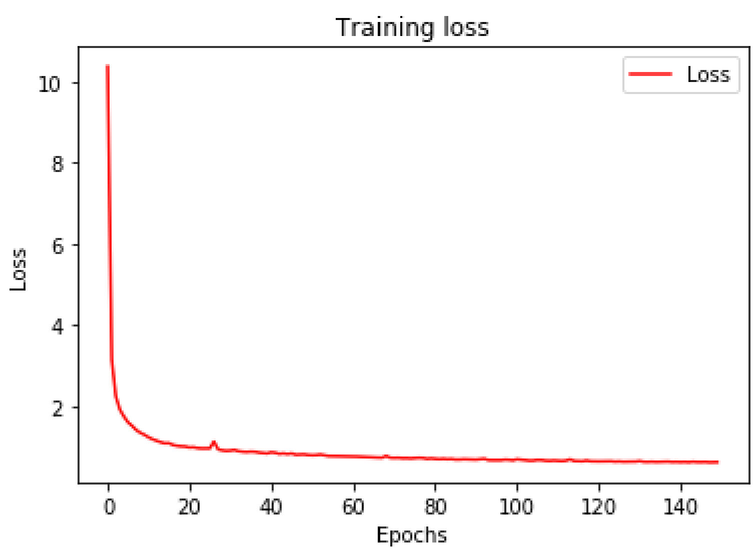
In [47]:
```
tf.keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy()
```

Out[47]: 0.8108871

In [49]:

```
import matplotlib.image  as mpimg
import matplotlib.pyplot as plt

#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
loss=history.history['loss']

epochs=range(len(loss)) # Get number of epochs


#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r')
plt.title('Training loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Loss"])

plt.figure()
```

Out[49]: <Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

Save Model

In [50]:

```
# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

Saved model to disk

In [ ]: