In this notebook I have used Adaptive Charging Network data (ACN) provided by California institute of technology to understand the user behaviour of EV owners. The dataset can be found at : [https://ev.caltech.edu/dataset (https://ev.caltech.edu/dataset)](https://ev.caltech.edu/dataset)

To prove my coding skills I have tried to code a part of the paper : ACN-Data : Analysis and Applications of an Open EV Charging Dataset which can be found at : [https://ev.caltech.edu/assets/pub/ACN_Data_Analysis_and_Applications.pdf (https://ev.caltech.edu/assets/pub/ACN_Data_Analysis_and_Applications.pdf)](https://ev.caltech.edu/assets/pub/ACN_Data_Analysis_and_Applications.pdf)

In [1]:
```
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from dateutil import tz
from datetime import datetime, timedelta

import tensorflow as tf
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Write a function that takes in a json file and converts it into a pandas DataFrame..
def json_to_DataFrame(file):
    with open(file) as data_file:
        data = json.load(data_file)
        df = pd.DataFrame(data["_items"])
        return df
```

In [3]:
```
caltech_df = json_to_DataFrame(file="acndata_sessions_caltech.json")
jpl_df = json_to_DataFrame(file="acndata_sessions_jpl.json")
```

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

Out[4]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5bc90cb9f9af8b0d7fe77cd2	0039	Wed, 25 Apr 2018 11:08:04 GMT	Wed, 25 Apr 2018 13:20:10 GMT	Wed, 25 Apr 2018 13:21:10 GMT	7.932	2_39_78_362_2018-04-25 11:08:04.400812	0002	CA-496	2-39-78-362	Americ
1	5bc90cb9f9af8b0d7fe77cd3	0039	Wed, 25 Apr 2018 13:45:10 GMT	Thu, 26 Apr 2018 00:56:16 GMT	Wed, 25 Apr 2018 16:44:15 GMT	10.013	2_39_95_27_2018-04-25 13:45:09.617470	0002	CA-319	2-39-95-27	Americ
2	5bc90cb9f9af8b0d7fe77cd4	0039	Wed, 25 Apr 2018 13:45:50 GMT	Wed, 25 Apr 2018 23:04:45 GMT	Wed, 25 Apr 2018 14:51:44 GMT	5.257	2_39_79_380_2018-04-25 13:45:49.962001	0002	CA-489	2-39-79-380	Americ
3	5bc90cb9f9af8b0d7fe77cd5	0039	Wed, 25 Apr 2018 14:37:06 GMT	Wed, 25 Apr 2018 23:55:34 GMT	Wed, 25 Apr 2018 16:05:22 GMT	5.177	2_39_79_379_2018-04-25 14:37:06.460772	0002	CA-327	2-39-79-379	Americ
4	5bc90cb9f9af8b0d7fe77cd6	0039	Wed, 25 Apr 2018 14:40:34 GMT	Wed, 25 Apr 2018 23:03:12 GMT	Wed, 25 Apr 2018 17:40:30 GMT	10.119	2_39_79_381_2018-04-25 14:40:33.638896	0002	CA-490	2-39-79-381	Americ

In [5]:
```
jpl_df.head()
```

Out[5]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5c36621bf9af8b4639a8e0b4	0001	Wed, 05 Sep 2018 11:04:13 GMT	Wed, 05 Sep 2018 19:09:35 GMT	None	9.583	1_1_179_800_2018-09-05 11:04:12.876087	0001	AG-3F32	1-1-179-800	Americ
1	5c36621bf9af8b4639a8e0b5	0001	Wed, 05 Sep 2018 11:08:09 GMT	Wed, 05 Sep 2018 14:09:02 GMT	None	7.114	1_1_179_794_2018-09-05 11:08:08.945820	0001	AG-3F20	1-1-179-794	Americ
2	5c36621bf9af8b4639a8e0b6	0001	Wed, 05 Sep 2018 12:35:14 GMT	Thu, 06 Sep 2018 00:30:12 GMT	None	11.774	1_1_179_797_2018-09-05 12:35:14.070250	0001	AG-3F23	1-1-179-797	Americ
3	5c36621bf9af8b4639a8e0b7	0001	Wed, 05 Sep 2018 12:51:31 GMT	Wed, 05 Sep 2018 22:32:58 GMT	None	6.280	1_1_179_781_2018-09-05 12:51:31.050539	0001	AG-3F31	1-1-179-781	Americ
4	5c36621bf9af8b4639a8e0b8	0001	Wed, 05 Sep 2018 13:08:28 GMT	Wed, 05 Sep 2018 23:32:52 GMT	None	7.022	1_1_179_787_2018-09-05 13:08:27.901538	0001	AG-3F16	1-1-179-787	Americ

In [6]:
```
# Checking if all the column names are same in both the DataFrames..
caltech_df.columns == jpl_df.columns
```

Out[6]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True])
```

In [7]:
```python
# Now we will compare the number of missing values and number of instances in each of the DataFrames..
df_info_dic = {"caltech_df":([caltech_df.shape[0]]+list(caltech_df.isnull().sum())),
               "jpl_df":([jpl_df.shape[0]]+list(jpl_df.isnull().sum()))}

df_info = pd.DataFrame(df_info_dic, index=["total_instances"]+list(caltech_df.columns))

df_info
```

Out[7]:

	caltech_df	jpl_df
total_instances	31424	33638
_id	0	0
clusterID	0	0
connectionTime	0	0
disconnectTime	0	0
doneChargingTime	2055	2037
kWhDelivered	0	0
sessionID	0	0
siteID	0	0
spaceID	0	0
stationID	0	0
timezone	0	0
userID	15036	2179
userInputs	15036	2179

Why there are so much missing values in caltech dataframe ?

Those drivers who use mobile application to input their Energy Demand and Estimated Departure Time are "claimed" drivers and those drivers who do not use the mobile application are "unclaimed" drivers. For unclaimed drivers Energy demand and estimated departure time are default values.

Caltech's EV Charger is open to both staff as well as public whereas JPL's EV Charger is open for staff only. It may happen, most of the public drivers are not aware of the mobile application, hence we see alot of missing values in userID column.

unclaimed drivers are charging their vehicles free of cost. Whereas claimed drivers are charging at $0.12 per KWh.

In [8]:
```python
# Let us fill in all the missing values in userID column as unclaimed.
caltech_df["userID"] = caltech_df["userID"].fillna("unclaimed")
jpl_df["userID"] = jpl_df["userID"].fillna("unclaimed")
```

In [9]:
```python
# Cross verify if all the missing values in userID columns are imputed or not..
caltech_df.userID.isnull().sum(), jpl_df.userID.isnull().sum()
```

Out[9]:
```
(0, 0)
```

In [10]:
```python
# Let us replace all other instances in userID column as claimed..

caltech_df.loc[caltech_df["userID"]!="unclaimed", "userID"]="claimed"
jpl_df.loc[jpl_df["userID"]!="unclaimed", "userID"]="claimed"
```

In [11]:
```python
caltech_df.head()
```

Out[11]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5bc90cb9f9af8b0d7fe77cd2	0039	Wed, 25 Apr 2018 11:08:04 GMT	Wed, 25 Apr 2018 13:20:10 GMT	Wed, 25 Apr 2018 13:21:10 GMT	7.932	2_39_78_362_2018-04-25 11:08:04.400812	0002	CA-496	2-39-78-362	Americ
1	5bc90cb9f9af8b0d7fe77cd3	0039	Wed, 25 Apr 2018 13:45:10 GMT	Thu, 26 Apr 2018 00:56:16 GMT	Wed, 25 Apr 2018 16:44:15 GMT	10.013	2_39_95_27_2018-04-25 13:45:09.617470	0002	CA-319	2-39-95-27	Americ
2	5bc90cb9f9af8b0d7fe77cd4	0039	Wed, 25 Apr 2018 13:45:50 GMT	Wed, 25 Apr 2018 23:04:45 GMT	Wed, 25 Apr 2018 14:51:44 GMT	5.257	2_39_79_380_2018-04-25 13:45:49.962001	0002	CA-489	2-39-79-380	Americ
3	5bc90cb9f9af8b0d7fe77cd5	0039	Wed, 25 Apr 2018 14:37:06 GMT	Wed, 25 Apr 2018 23:55:34 GMT	Wed, 25 Apr 2018 16:05:22 GMT	5.177	2_39_79_379_2018-04-25 14:37:06.460772	0002	CA-327	2-39-79-379	Americ
4	5bc90cb9f9af8b0d7fe77cd6	0039	Wed, 25 Apr 2018 14:40:34 GMT	Wed, 25 Apr 2018 23:03:12 GMT	Wed, 25 Apr 2018 17:40:30 GMT	10.119	2_39_79_381_2018-04-25 14:40:33.638896	0002	CA-490	2-39-79-381	Americ

In [12]:
```python
caltech_df.userID.value_counts() # Number of unclaimed should be equal to number of missing values in the column.
```

Out[12]:
```
claimed      16388
unclaimed    15036
Name: userID, dtype: int64
```

In [13]:
```
jpl_df.userID.value_counts()
```

Out[13]:
```
claimed      31459
unclaimed     2179
Name: userID, dtype: int64
```

In [14]:
```
df_info_dic = {"caltech_df":([caltech_df.shape[0]]+list(caltech_df.isnull().sum())),
               "jpl_df":([jpl_df.shape[0]]+list(jpl_df.isnull().sum()))}

df_info = pd.DataFrame(df_info_dic, index=["total_instances"]+list(caltech_df.columns))
df_info
```

Out[14]:

	caltech_df	jpl_df
total_instances	31424	33638
_id	0	0
clusterID	0	0
connectionTime	0	0
disconnectTime	0	0
doneChargingTime	2055	2037
kWhDelivered	0	0
sessionID	0	0
siteID	0	0
spaceID	0	0
stationID	0	0
timezone	0	0
userID	0	0
userInputs	15036	2179

Check clusterID column of each dataframes

In [15]:
```
caltech_df["clusterID"].value_counts()
```

Out[15]:
```
0039    29343
39       2081
Name: clusterID, dtype: int64
```

In [16]:
```
jpl_df["clusterID"].value_counts()
```

Out[16]:
```
0001    33638
Name: clusterID, dtype: int64
```

In [17]:
```
# Let us change the clusterID of each row for caltech_df to "0039"
caltech_df["clusterID"]="0039"
```

In [18]:
```
print(caltech_df["clusterID"].value_counts())
print(jpl_df["clusterID"].value_counts())
```

```
0039    31424
Name: clusterID, dtype: int64
0001    33638
Name: clusterID, dtype: int64
```

Check the siteID of each dataframe

In [19]:
```
print(caltech_df["siteID"].value_counts())
print(jpl_df["siteID"].value_counts())
```

```
0002    29343
2        2081
Name: siteID, dtype: int64
0001    33638
Name: siteID, dtype: int64
```

In [20]:
```
# Let us change the siteID of each row of caltech_df to "0002"
caltech_df["siteID"]="0002"
```

In [21]:
```
caltech_df["siteID"].value_counts()
```

Out[21]:
```
0002    31424
Name: siteID, dtype: int64
```

Check the spaceID of each DataFrame.

In [22]:
```
print("Number of chargers in Caltech  :  ", len(caltech_df["spaceID"].value_counts()))
print("Number of chargers in JPL      :  ", len(jpl_df["spaceID"].value_counts()))
```

```
Number of chargers in Caltech  :   55
Number of chargers in JPL      :   52
```

Check the timezone column of each dataframe.

In [23]:
```
print("timezone caltech     :    ",caltech_df["timezone"].value_counts())
print("\ntimezone jpl         :    ",jpl_df["timezone"].value_counts())
```

```
timezone caltech     :     America/Los_Angeles    31424
Name: timezone, dtype: int64

timezone jpl         :     America/Los_Angeles    33638
Name: timezone, dtype: int64
```

Note : Though the timezone column has America/Los_Angeles is given as TimeZone. But the connection times and disconnect times are given in GMT. So we have to convert them to America/Los_Angeles timezone.

Convert connectionTime, disconnectTime and doneChargingTime columns into a datetime

- connectionTime : It is that time when the EV was plugged in.
- disconnectTime : It is that time when the EV was unplugged.
- doneChargingTime : It is that time when the last non-zero current draw was recorded. It means it is that time when the EV's battery became fully charged. So let us fill all the missing values of this column with the corresponding values of disconnectTime column.

In [24]:
```
caltech_df["doneChargingTime"].isnull().sum()
```

Out[24]:
```
2055
```

In [25]:
```
jpl_df["doneChargingTime"].isnull().sum()
```

Out[25]:
```
2037
```

In [26]:
```
caltech_df["doneChargingTime"] = caltech_df["doneChargingTime"].fillna(caltech_df["disconnectTime"])
```

In [27]:
```
caltech_df["doneChargingTime"].isnull().sum()
```
Out[27]:
```
0
```
In [28]:
```
jpl_df["doneChargingTime"] = jpl_df["doneChargingTime"].fillna(jpl_df["disconnectTime"])
```
In [29]:
```
jpl_df["doneChargingTime"].isnull().sum()
```
Out[29]:
```
0
```
In [30]:
```
# Now we will convert each instances of connectionTime, disconnectTime and doneChargingTime in datetime objects. Also
# we will convert the timezone from UTC to America/Los_Angeles Timezone.
def convert_datetime(df):
    from_zone = tz.gettz('UTC')
    to_zone = tz.gettz('America/Los_Angeles')

    df.iloc[:, 2] = df.iloc[:, 2].apply(lambda x : datetime.strptime(x, "%a, %d %b %Y %H:%M:%S %Z"))
    df.iloc[:, 3] = df.iloc[:, 3].apply(lambda x : datetime.strptime(x, "%a, %d %b %Y %H:%M:%S %Z"))
    df.iloc[:, 4] = df.iloc[:, 4].apply(lambda x : datetime.strptime(x, "%a, %d %b %Y %H:%M:%S %Z"))

    df.iloc[:,2] = df.iloc[:, 2].apply(lambda x : x.replace(tzinfo=from_zone))
    df.iloc[:,3] = df.iloc[:, 3].apply(lambda x : x.replace(tzinfo=from_zone))
    df.iloc[:,4] = df.iloc[:, 4].apply(lambda x : x.replace(tzinfo=from_zone))

    df.iloc[:,2] = df.iloc[:, 2].apply(lambda x : x.astimezone(to_zone))
    df.iloc[:,3] = df.iloc[:, 3].apply(lambda x : x.astimezone(to_zone))
    df.iloc[:,4] = df.iloc[:, 4].apply(lambda x : x.astimezone(to_zone))

    return df
```
In [31]:
```
caltech_df = convert_datetime(caltech_df)
jpl_df = convert_datetime(jpl_df)
```

```
In [32]: caltech_df.head()
```

Out[32]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5bc90cb9f9af8b0d7fe77cd2	0039	2018-04-25 04:08:04-07:00	2018-04-25 06:20:10-07:00	2018-04-25 06:21:10-07:00	7.932	2_39_78_362_2018-04-25 11:08:04.400812	0002	CA-496	2-39-78-362	Americ
1	5bc90cb9f9af8b0d7fe77cd3	0039	2018-04-25 06:45:10-07:00	2018-04-25 17:56:16-07:00	2018-04-25 09:44:15-07:00	10.013	2_39_95_27_2018-04-25 13:45:09.617470	0002	CA-319	2-39-95-27	Americ
2	5bc90cb9f9af8b0d7fe77cd4	0039	2018-04-25 06:45:50-07:00	2018-04-25 16:04:45-07:00	2018-04-25 07:51:44-07:00	5.257	2_39_79_380_2018-04-25 13:45:49.962001	0002	CA-489	2-39-79-380	Americ
3	5bc90cb9f9af8b0d7fe77cd5	0039	2018-04-25 07:37:06-07:00	2018-04-25 16:55:34-07:00	2018-04-25 09:05:22-07:00	5.177	2_39_79_379_2018-04-25 14:37:06.460772	0002	CA-327	2-39-79-379	Americ
4	5bc90cb9f9af8b0d7fe77cd6	0039	2018-04-25 07:40:34-07:00	2018-04-25 16:03:12-07:00	2018-04-25 10:40:30-07:00	10.119	2_39_79_381_2018-04-25 14:40:33.638896	0002	CA-490	2-39-79-381	Americ

In [33]:
```
jpl_df.head()
```
Out[33]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5c36621bf9af8b4639a8e0b4	0001	2018-09-05 04:04:13-07:00	2018-09-05 12:09:35-07:00	2018-09-05 12:09:35-07:00	9.583	1_1_179_800_2018-09-05 11:04:12.876087	0001	AG-3F32	1-1-179-800	Americ
1	5c36621bf9af8b4639a8e0b5	0001	2018-09-05 04:08:09-07:00	2018-09-05 07:09:02-07:00	2018-09-05 07:09:02-07:00	7.114	1_1_179_794_2018-09-05 11:08:08.945820	0001	AG-3F20	1-1-179-794	Americ
2	5c36621bf9af8b4639a8e0b6	0001	2018-09-05 05:35:14-07:00	2018-09-05 17:30:12-07:00	2018-09-05 17:30:12-07:00	11.774	1_1_179_797_2018-09-05 12:35:14.070250	0001	AG-3F23	1-1-179-797	Americ
3	5c36621bf9af8b4639a8e0b7	0001	2018-09-05 05:51:31-07:00	2018-09-05 15:32:58-07:00	2018-09-05 15:32:58-07:00	6.280	1_1_179_781_2018-09-05 12:51:31.050539	0001	AG-3F31	1-1-179-781	Americ
4	5c36621bf9af8b4639a8e0b8	0001	2018-09-05 06:08:28-07:00	2018-09-05 16:32:52-07:00	2018-09-05 16:32:52-07:00	7.022	1_1_179_787_2018-09-05 13:08:27.901538	0001	AG-3F16	1-1-179-787	Americ

Adding session_duration column

In [34]:
```
caltech_df["session_duration"] = (caltech_df["disconnectTime"] - caltech_df["connectionTime"])/timedelta(minutes=1)
```

In [35]:
```
jpl_df["session_duration"] = (jpl_df["disconnectTime"] - jpl_df["connectionTime"])/timedelta(minutes=1)
```

In [36]:

```
caltech_df.head()
```

Out[36]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5bc90cb9f9af8b0d7fe77cd2	0039	2018-04-25 04:08:04-07:00	2018-04-25 06:20:10-07:00	2018-04-25 06:21:10-07:00	7.932	2_39_78_362_2018-04-25 11:08:04.400812	0002	CA-496	2-39-78-362	Americ
1	5bc90cb9f9af8b0d7fe77cd3	0039	2018-04-25 06:45:10-07:00	2018-04-25 17:56:16-07:00	2018-04-25 09:44:15-07:00	10.013	2_39_95_27_2018-04-25 13:45:09.617470	0002	CA-319	2-39-95-27	Americ
2	5bc90cb9f9af8b0d7fe77cd4	0039	2018-04-25 06:45:50-07:00	2018-04-25 16:04:45-07:00	2018-04-25 07:51:44-07:00	5.257	2_39_79_380_2018-04-25 13:45:49.962001	0002	CA-489	2-39-79-380	Americ
3	5bc90cb9f9af8b0d7fe77cd5	0039	2018-04-25 07:37:06-07:00	2018-04-25 16:55:34-07:00	2018-04-25 09:05:22-07:00	5.177	2_39_79_379_2018-04-25 14:37:06.460772	0002	CA-327	2-39-79-379	Americ
4	5bc90cb9f9af8b0d7fe77cd6	0039	2018-04-25 07:40:34-07:00	2018-04-25 16:03:12-07:00	2018-04-25 10:40:30-07:00	10.119	2_39_79_381_2018-04-25 14:40:33.638896	0002	CA-490	2-39-79-381	Americ

In [37]:
```python
jpl_df.head()
```

Out[37]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5c36621bf9af8b4639a8e0b4	0001	2018-09-05 04:04:13-07:00	2018-09-05 12:09:35-07:00	2018-09-05 12:09:35-07:00	9.583	1_1_179_800_2018-09-05 11:04:12.876087	0001	AG-3F32	1-1-179-800	Americ
1	5c36621bf9af8b4639a8e0b5	0001	2018-09-05 04:08:09-07:00	2018-09-05 07:09:02-07:00	2018-09-05 07:09:02-07:00	7.114	1_1_179_794_2018-09-05 11:08:08.945820	0001	AG-3F20	1-1-179-794	Americ
2	5c36621bf9af8b4639a8e0b6	0001	2018-09-05 05:35:14-07:00	2018-09-05 17:30:12-07:00	2018-09-05 17:30:12-07:00	11.774	1_1_179_797_2018-09-05 12:35:14.070250	0001	AG-3F23	1-1-179-797	Americ
3	5c36621bf9af8b4639a8e0b7	0001	2018-09-05 05:51:31-07:00	2018-09-05 15:32:58-07:00	2018-09-05 15:32:58-07:00	6.280	1_1_179_781_2018-09-05 12:51:31.050539	0001	AG-3F31	1-1-179-781	Americ
4	5c36621bf9af8b4639a8e0b8	0001	2018-09-05 06:08:28-07:00	2018-09-05 16:32:52-07:00	2018-09-05 16:32:52-07:00	7.022	1_1_179_787_2018-09-05 13:08:27.901538	0001	AG-3F16	1-1-179-787	Americ

Adding a Day column to both the DataFrames that signifies whether the EV was charged on a weekDay or a weekEnd

In [38]:
```python
caltech_df["Day"] = caltech_df["connectionTime"].apply(lambda x : x.strftime("%a"))
caltech_df["Day"] = caltech_df["Day"].apply(lambda x : "weekEnd" if (x=="Sun" or x=="Sat") else "weekDay")
```

In [39]:
```python
jpl_df["Day"] = jpl_df["connectionTime"].apply(lambda x : x.strftime("%a"))
jpl_df["Day"] = jpl_df["Day"].apply(lambda x : "weekEnd" if (x=="Sun" or x=="Sat") else "weekDay")
```

In [40]:
```python
# Let us check the number of vehicles charged on weekDays compared to weekEnds..
caltech_df["Day"].value_counts(normalize=True)
```

Out[40]:
```
weekDay    0.846996
weekEnd    0.153004
Name: Day, dtype: float64
```

In [41]:
```
jpl_df["Day"].value_counts(normalize=True)
```

Out[41]:
```
weekDay    0.97369
weekEnd    0.02631
Name: Day, dtype: float64
```

Note :

- EV Charger installed in JPL is for employees only. This is the reason why only 2.6% vehicles are charging on weekEnds.
- Whereas the Caltech EV Charger is open for outsiders also hence we can see a significant number of vehicles are charging on weekEnds here.

TimeSeries analysis of each DataFrame

In [42]:
```
caltech_ts = caltech_df[["kWhDelivered"]]
caltech_ts.index = caltech_df["connectionTime"]
```

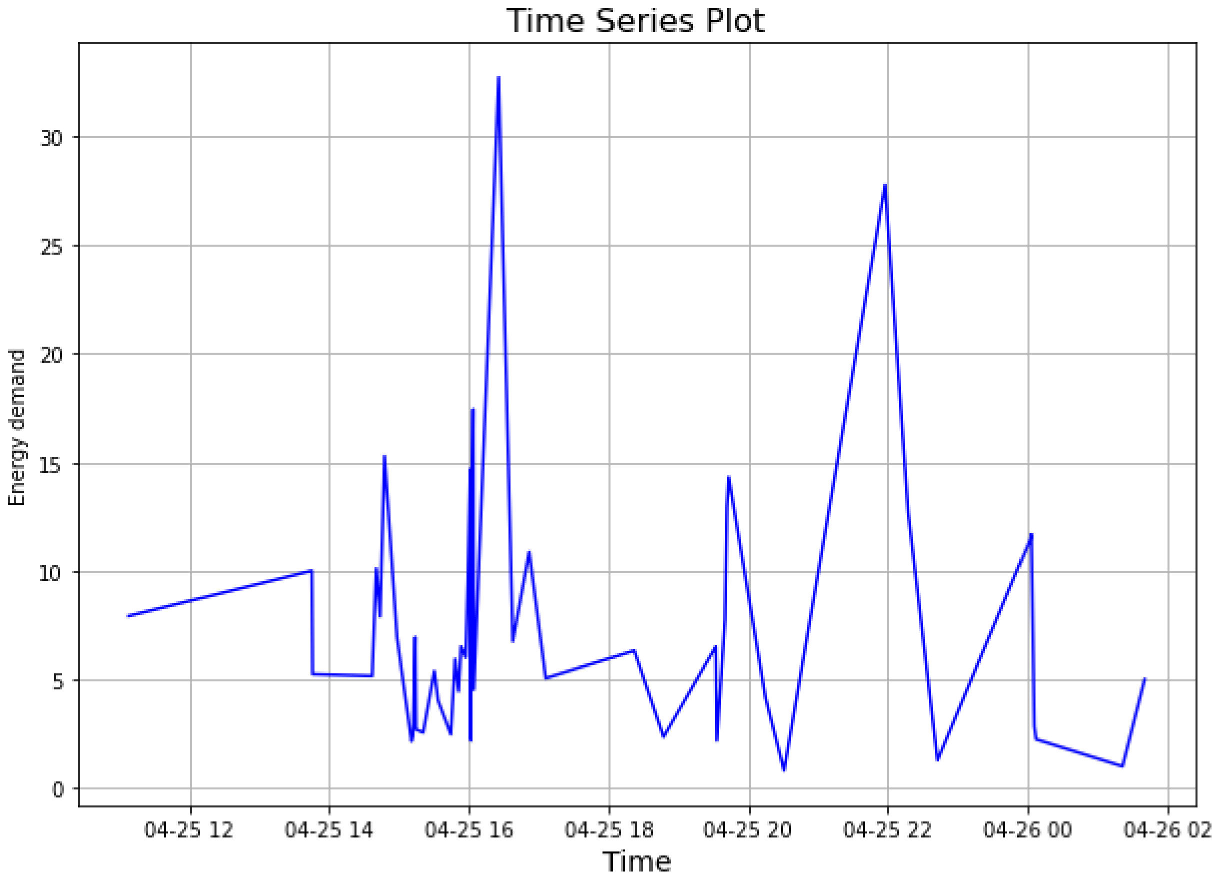
In [43]:
```
caltech_ts.head()
```

Out[43]:

	kWhDelivered
connectionTime	
2018-04-25 04:08:04-07:00	7.932
2018-04-25 06:45:10-07:00	10.013
2018-04-25 06:45:50-07:00	5.257
2018-04-25 07:37:06-07:00	5.177
2018-04-25 07:40:34-07:00	10.119

In [44]:
```python
# Now let us make a function to plot time series plots..
def plot_time_series(df, start=0, end=None, font_size=14, title_font_size=16,label=None, color="b"):
    plt.plot(df[start:end], label=label, c=color)
    plt.title("Time Series Plot", fontsize=title_font_size)
    if label:
        plt.legend(fontsize=font_size)
    plt.xlabel("Time", fontsize=font_size)
    plt.ylabel("Energy demand")
    plt.grid()
    plt.show()
```

In [45]:
```python
plt.figure(figsize=(10,7))
plot_time_series(caltech_ts[:50])

# The plot doesn't look like a time series plot.
```



In [46]:
```
# Let us add a column as connectionDate in each dataframe..
caltech_df["connectionDate"] = caltech_df["connectionTime"].apply(lambda x : x.date)
jpl_df["connectionDate"] = jpl_df["connectionTime"].apply(lambda x : x.date)
```

In [47]:
```
caltech_df.head()
```

Out[47]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5bc90cb9f9af8b0d7fe77cd2	0039	2018-04-25 04:08:04-07:00	2018-04-25 06:20:10-07:00	2018-04-25 06:21:10-07:00	7.932	2_39_78_362_2018-04-25 11:08:04.400812	0002	CA-496	2-39-78-362	Americ
1	5bc90cb9f9af8b0d7fe77cd3	0039	2018-04-25 06:45:10-07:00	2018-04-25 17:56:16-07:00	2018-04-25 09:44:15-07:00	10.013	2_39_95_27_2018-04-25 13:45:09.617470	0002	CA-319	2-39-95-27	Americ
2	5bc90cb9f9af8b0d7fe77cd4	0039	2018-04-25 06:45:50-07:00	2018-04-25 16:04:45-07:00	2018-04-25 07:51:44-07:00	5.257	2_39_79_380_2018-04-25 13:45:49.962001	0002	CA-489	2-39-79-380	Americ
3	5bc90cb9f9af8b0d7fe77cd5	0039	2018-04-25 07:37:06-07:00	2018-04-25 16:55:34-07:00	2018-04-25 09:05:22-07:00	5.177	2_39_79_379_2018-04-25 14:37:06.460772	0002	CA-327	2-39-79-379	Americ
4	5bc90cb9f9af8b0d7fe77cd6	0039	2018-04-25 07:40:34-07:00	2018-04-25 16:03:12-07:00	2018-04-25 10:40:30-07:00	10.119	2_39_79_381_2018-04-25 14:40:33.638896	0002	CA-490	2-39-79-381	Americ

In [48]:
```
jpl_df.head()
```

Out[48]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5c36621bf9af8b4639a8e0b4	0001	2018-09-05 04:04:13-07:00	2018-09-05 12:09:35-07:00	2018-09-05 12:09:35-07:00	9.583	1_1_179_800_2018-09-05 11:04:12.876087	0001	AG-3F32	1-1-179-800	Americ
1	5c36621bf9af8b4639a8e0b5	0001	2018-09-05 04:08:09-07:00	2018-09-05 07:09:02-07:00	2018-09-05 07:09:02-07:00	7.114	1_1_179_794_2018-09-05 11:08:08.945820	0001	AG-3F20	1-1-179-794	Americ
2	5c36621bf9af8b4639a8e0b6	0001	2018-09-05 05:35:14-07:00	2018-09-05 17:30:12-07:00	2018-09-05 17:30:12-07:00	11.774	1_1_179_797_2018-09-05 12:35:14.070250	0001	AG-3F23	1-1-179-797	Americ
3	5c36621bf9af8b4639a8e0b7	0001	2018-09-05 05:51:31-07:00	2018-09-05 15:32:58-07:00	2018-09-05 15:32:58-07:00	6.280	1_1_179_781_2018-09-05 12:51:31.050539	0001	AG-3F31	1-1-179-781	Americ
4	5c36621bf9af8b4639a8e0b8	0001	2018-09-05 06:08:28-07:00	2018-09-05 16:32:52-07:00	2018-09-05 16:32:52-07:00	7.022	1_1_179_787_2018-09-05 13:08:27.901538	0001	AG-3F16	1-1-179-787	Americ

Now let a make a function that adds two columns in the dataframe and those are total_energy_consumed and total_sessions per day.

The following steps should be performed.

- collect all the instances in connectionDate column in a list that is list1.
- Remove the duplicate elements and store the unique elements in list2 and sort it so as we get the dates in a proper fashion.
- Find the indices of these unique elements in list2 from list1.
- Now count the duplicate values in list1. From this we will find the sessions served each day.
- With the help of list of indices and duplicate values, we will find the energy demand each day.
- Then we will make a dictionary of connectionDate, energyDemand and sessions served each day.
- Convert this dictionary into a DataFrame.
- Make connectionDate as the index of the DataFrame
- Finally return the DataFrame.

In [49]:
```
# Now let us make a function that calculates total energy consumed and total sessions served on a single day.
def make_correct_time_series(df):
    list1 = list(df["connectionDate"])
    list2 = list(set(list1))
    list2.sort()

    indices_list = []
    for i in list2:
        indices_list.append(list1.index(i))

    sessions_served_each_day = []
    for i in list2:
        sessions_served_each_day.append(list1.count(i))

    # Calculate energy demand on a specific day..
    energy_demand_per_day = []
    for i, j in zip(indices_list, sessions_served_each_day):
        energy_demand = []
        for x in df.iloc[i:(i+j), 5]:
            energy_demand.append(x)
        energy_demand_per_day.append(np.round(np.sum(np.array(energy_demand)),2))

    # Now make a dict of connectionDate, energyDemand and sessions.
    df_dic = {"connectionDate":list2,
              "energyDemand" : energy_demand_per_day,
              "sessions" : sessions_served_each_day}

    # Now convert this dictionary into a DataFrame.
    ts = pd.DataFrame(df_dic)
    # make connectionDate as the index of the DataFrame.
    ts = ts.set_index(["connectionDate"])

    return ts
```

In [50]:
```
caltech_ts = make_correct_time_series(caltech_df)
jpl_ts = make_correct_time_series(jpl_df)
```

In [51]:
```
caltech_ts.head()
```

Out[51]:

	energyDemand	sessions
connectionDate		
2018-04-25	447.94	60
2018-04-26	338.81	47
2018-04-27	572.43	51
2018-04-28	341.14	30
2018-04-29	266.26	29

In [52]:
```
jpl_ts.head()
```

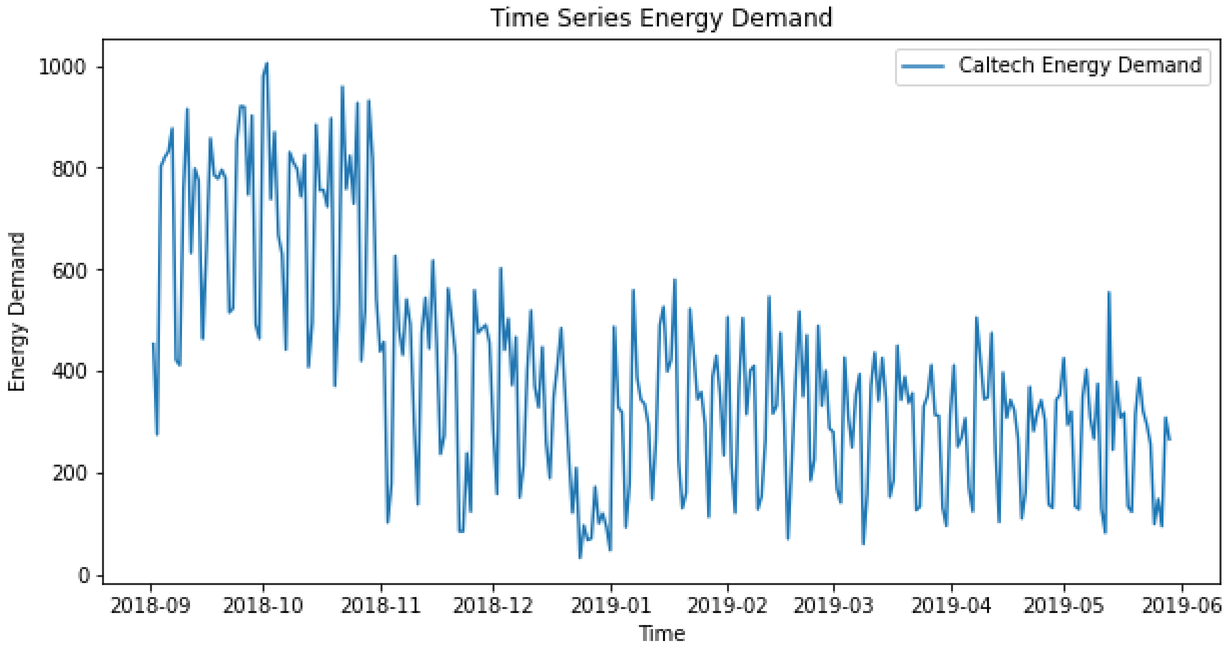
Out[52]:

	energyDemand	sessions
connectionDate		
2018-09-05	262.37	30
2018-09-06	428.49	44
2018-09-07	471.46	47
2018-09-08	36.61	4
2018-09-09	22.44	2

Understanding User Behaviour

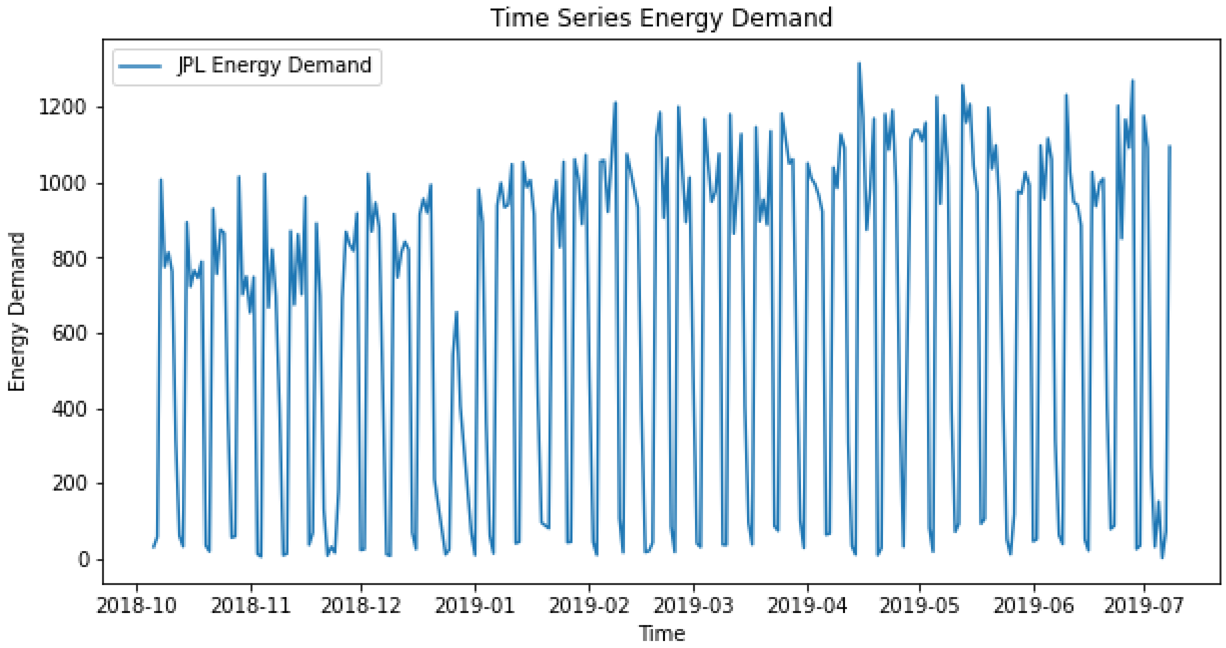
In [53]:
```python
plt.figure(figsize=(10,5))
plt.plot(caltech_ts["energyDemand"][130:400], label="Caltech Energy Demand")
plt.title("Time Series Energy Demand")
plt.xlabel("Time")
plt.ylabel("Energy Demand")
plt.legend()
```

Out[53]: <matplotlib.legend.Legend at 0x1c34b48a3a0>



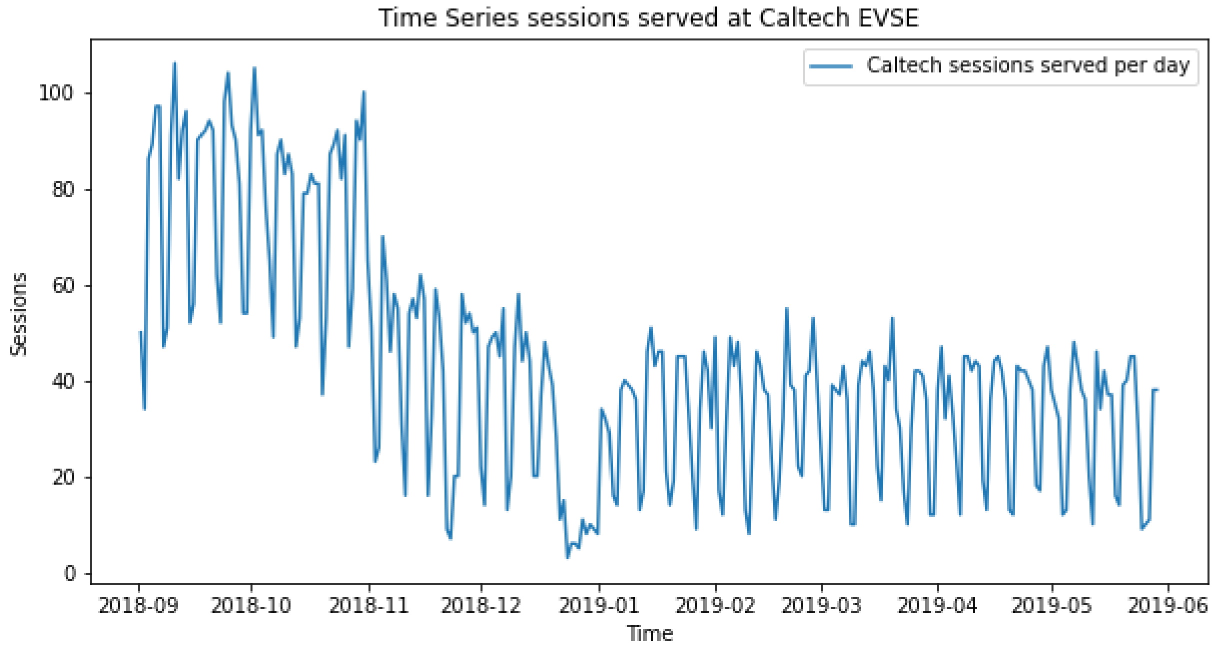
In [54]:
```python
plt.figure(figsize=(10,5))
plt.plot(jpl_ts["energyDemand"][30:300], label="JPL Energy Demand")
plt.title("Time Series Energy Demand")
plt.xlabel("Time")
plt.ylabel("Energy Demand")
plt.legend()
```

Out[54]: <matplotlib.legend.Legend at 0x1c34b559220>



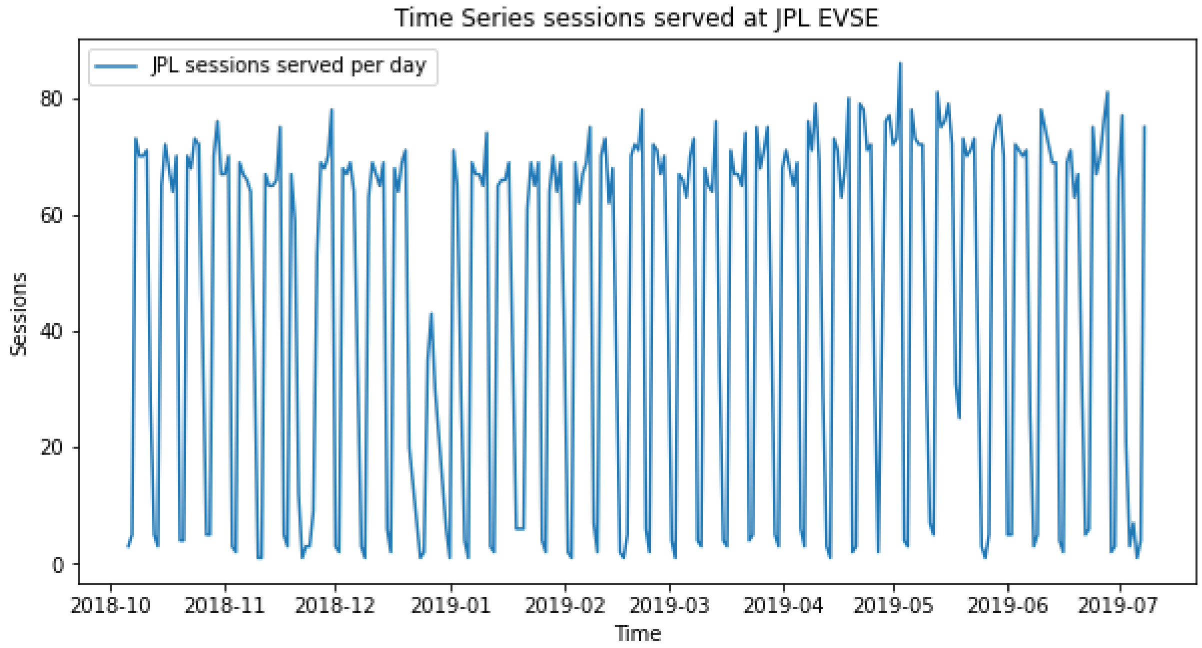
In [55]:
```
plt.figure(figsize=(10,5))
plt.plot(caltech_ts["sessions"][130:400], label="Caltech sessions served per day")
plt.title("Time Series sessions served at Caltech EVSE")
plt.xlabel("Time")
plt.ylabel("Sessions")
plt.legend()
```

Out[55]: <matplotlib.legend.Legend at 0x1c34b5b3760>



In [56]:
```
plt.figure(figsize=(10,5))
plt.plot(jpl_ts["sessions"][30:300], label="JPL sessions served per day")
plt.title("Time Series sessions served at JPL EVSE")
plt.xlabel("Time")
plt.ylabel("Sessions")
plt.legend()
```

Out[56]: <matplotlib.legend.Legend at 0x1c34b5f5580>



- ***For Caltech ACN :*** The data confirms the difference between paid and free charging facilities. During the first 2.5 years of operation the Caltech ACN was free for drivers. However, from November 1, 2018 a fee of 12 cents per kWh was imposed. We can see this date clearly in the above figure. Right after November 1, 2018 there is a sudden drop in energy demand as well as the number of sessions per day.
- ***For JPL ACN :*** Because of an issue with the site configuration, approximately half of the charging stations at JPL were free prior to November 1, 2018. After this date, the same fee was also imposed here but we do not see any similar fall in charging demand here because the demand for charging here is high. This high demand overshadows any price sensitivity.

Analysing connectionTime and disconnectTime columns

In [57]:
```
caltech_new = caltech_df.copy()
jpl_new = jpl_df.copy()
```

In [58]:
```
caltech_new["connectionTime"] = caltech_new["connectionTime"].apply(lambda x: np.round(x.time().hour + (x.time().minute)/60))
caltech_new["disconnectTime"] = caltech_new["disconnectTime"].apply(lambda x: np.round(x.time().hour + (x.time().minute)/60))
caltech_new["doneChargingTime"] = caltech_new["doneChargingTime"].apply(lambda x: np.round(x.time().hour + (x.time().minute)/60))
```

In [59]:
```
jpl_new["connectionTime"] = jpl_new["connectionTime"].apply(lambda x: np.round(x.time().hour + (x.time().minute)/60))
jpl_new["disconnectTime"] = jpl_new["disconnectTime"].apply(lambda x: np.round(x.time().hour + (x.time().minute)/60))
jpl_new["doneChargingTime"] = jpl_new["doneChargingTime"].apply(lambda x: np.round(x.time().hour + (x.time().minute)/60))
```

In [60]:
```
caltech_new["connectionMonth"] = caltech_new["connectionDate"].apply(lambda x : x.strftime("%b"))
```

In [61]:
```
caltech_new.head()
```

Out[61]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5bc90cb9f9af8b0d7fe77cd2	0039	4.0	6.0	6.0	7.932	2_39_78_362_2018-04-25 11:08:04.400812	0002	CA-496	2-39-78-362	Americ
1	5bc90cb9f9af8b0d7fe77cd3	0039	7.0	18.0	10.0	10.013	2_39_95_27_2018-04-25 13:45:09.617470	0002	CA-319	2-39-95-27	Americ
2	5bc90cb9f9af8b0d7fe77cd4	0039	7.0	16.0	8.0	5.257	2_39_79_380_2018-04-25 13:45:49.962001	0002	CA-489	2-39-79-380	Americ
3	5bc90cb9f9af8b0d7fe77cd5	0039	8.0	17.0	9.0	5.177	2_39_79_379_2018-04-25 14:37:06.460772	0002	CA-327	2-39-79-379	Americ
4	5bc90cb9f9af8b0d7fe77cd6	0039	8.0	16.0	11.0	10.119	2_39_79_381_2018-04-25 14:40:33.638896	0002	CA-490	2-39-79-381	Americ

In [62]:
```
jpl_new["connectionMonth"] = jpl_new["connectionDate"].apply(lambda x : x.strftime("%b"))
```

In [63]:
```
# Plotting arrival time for Free and Paid users on weekDays and weekEnds

a1 = caltech_new.loc[(caltech_new["userID"]=="unclaimed")&(caltech_new["Day"]=="weekDay")]["connectionTime"]
a2 = caltech_new.loc[(caltech_new["userID"]=="claimed")&(caltech_new["Day"]=="weekDay")]["connectionTime"]
a3 = caltech_new.loc[(caltech_new["userID"]=="unclaimed")&(caltech_new["Day"]=="weekEnd")]["connectionTime"]
a4 = caltech_new.loc[(caltech_new["userID"]=="claimed")&(caltech_new["Day"]=="weekEnd")]["connectionTime"]
```

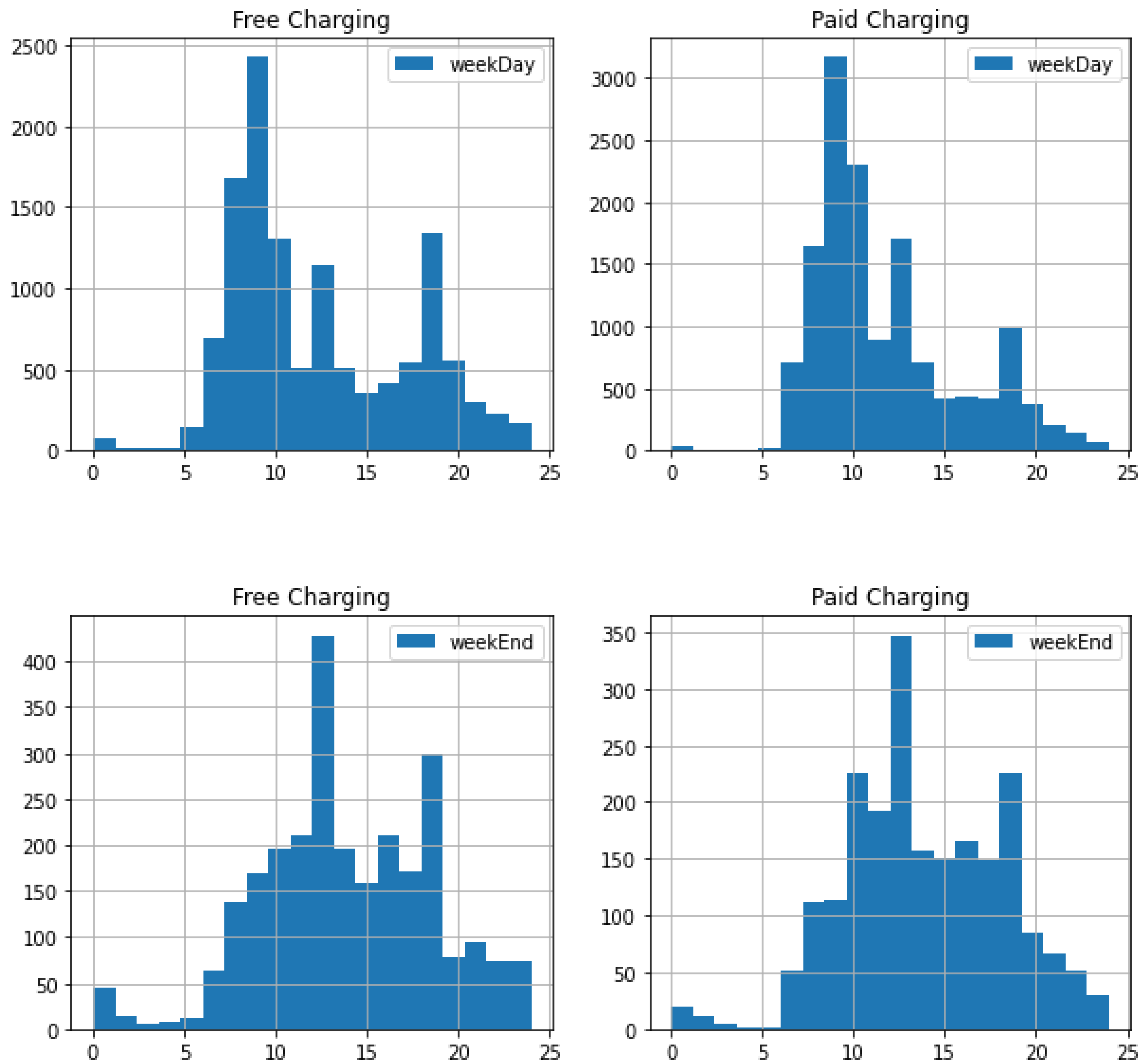
In [64]:

```python
fig, axes = plt.subplots(2,2, figsize=(10,10))

fig.subplots_adjust(hspace=0.4, top=0.85)
fig.suptitle("Arrival Time Analysis for Paid and Free Users on weekDays and weekEnds", fontsize=16)

a1.hist(bins=20, ax=axes[0][0], label="weekDay")
a2.hist(bins=20, ax=axes[0][1], label="weekDay")
axes[0][0].set_title("Free Charging")
axes[0][1].set_title("Paid Charging")
axes[0][0].legend()
axes[0][1].legend()

a3.hist(bins=20, ax=axes[1][0], label="weekEnd")
a4.hist(bins=20, ax=axes[1][1], label="weekEnd")
axes[1][0].set_title("Free Charging")
axes[1][1].set_title("Paid Charging")
axes[1][0].legend()
axes[1][1].legend()
```

Out[64]:

```
<matplotlib.legend.Legend at 0x1c34b797b50>
```

Arrival Time Analysis for Paid and Free Users on weekDays and weekEnds



In [65]:

```
# Plotting departure time for Free and Paid users on weekDays and weekEnds for Caltech EVSE

d1 = caltech_new.loc[(caltech_new["userID"]=="unclaimed")&(caltech_new["Day"]=="weekDay")]["disconnectTime"]
d2 = caltech_new.loc[(caltech_new["userID"]=="claimed")&(caltech_new["Day"]=="weekDay")]["disconnectTime"]
d3 = caltech_new.loc[(caltech_new["userID"]=="unclaimed")&(caltech_new["Day"]=="weekEnd")]["disconnectTime"]
d4 = caltech_new.loc[(caltech_new["userID"]=="claimed")&(caltech_new["Day"]=="weekEnd")]["disconnectTime"]
```

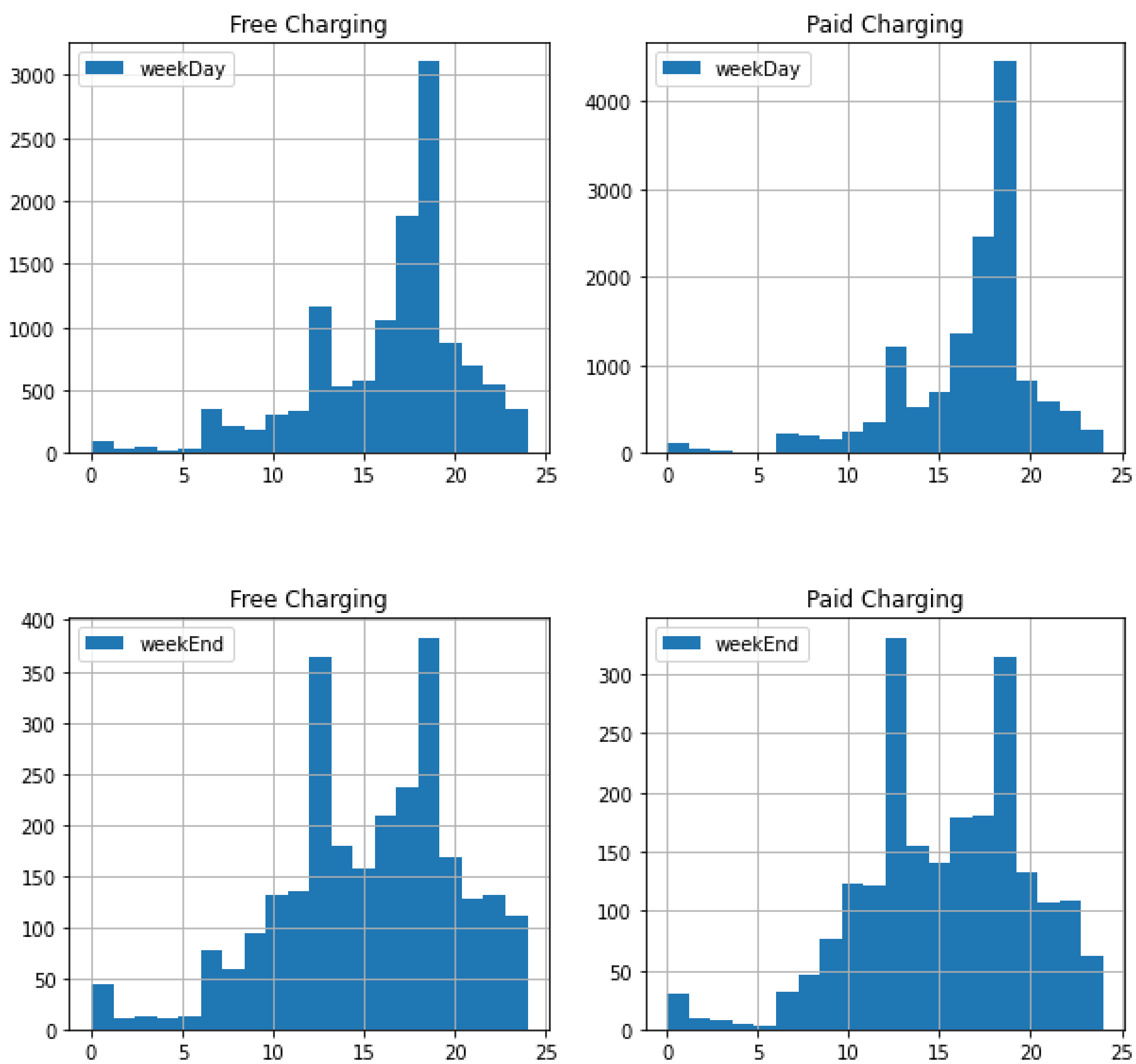
In [66]:

```
fig, axes = plt.subplots(2,2, figsize=(10,10))
fig.subplots_adjust(hspace=0.4, top=0.85)
fig.suptitle("Departure Time Analysis for Paid and Free Users on weekDays and weekEnds", fontsize=16)
d1.hist(bins=20, ax=axes[0][0], label="weekDay")
d2.hist(bins=20, ax=axes[0][1], label="weekDay")
axes[0][0].set_title("Free Charging")
axes[0][1].set_title("Paid Charging")
axes[0][0].legend()
axes[0][1].legend()

d3.hist(bins=20, ax=axes[1][0], label="weekEnd")
d4.hist(bins=20, ax=axes[1][1], label="weekEnd")
axes[1][0].set_title("Free Charging")
axes[1][1].set_title("Paid Charging")
axes[1][0].legend()
axes[1][1].legend()
```

Out[66]:

```
<matplotlib.legend.Legend at 0x1c34bd95880>
```

Departure Time Analysis for Paid and Free Users on weekDays and weekEnds



In [ ]:

Effect of Free vs Paid charging in Caltech EVSE

1. Arrival time analysis

weekDay distribution

- From the figure we can conclude that the shape of the distributions are similar before and after paid charging was implemented.
- However two key differences have been observed in weekDay charging between free and paid charging.
    - First : The peak around 6 pm vanishes as paid charging was implemented. This shows there is a decrement in the community usage of the Caltech ACN after its cost became comparable to at-home charging. So people are preferring to charge their EVs at home instead of standing in a queue at caltech charging station.
    - Second : The peak in arrivals at around 8 am increases. This may happen because those who are not charging thier EVs in the evening may be using the station in the morning (after coming to office they are connecting their EVs to the charging station.
- On weekDays there is a morning peak.This means people may be queuing to wait for their chance to pluggin their vehicle. This necessitates a larger infrastucture capacity in the future. As the demand is high in the morning, the owners of the EV station can increase the charging fees in the morning to compensate the infrastructure increment cost.

weekEnd distribution

- Since the caltech ACN is open to the public and is located on a university campus , it receives the users on the weekEnds too.
- We can see a peak at noon for both paid and free. But the peak for paid is lower in comparision to free beacause people perhaps prefer to charge their EVs at home.

Model Building

- Here I want to build a model to predict Energy Delivered based on features such as connectionTime, session Duration etc.
- Since our target variable is a continuous value hence we have to build a regression model.
- A model has been built in the past using this dataset where the author has predicted Charging demand at public charging stations using XGBoost machine learning method and has achieved R-squared value of 0.52, Mean Absolute Error of 4.6 kWh.
- **The link of the paper is :** https://www.researchgate.net/publication/343693033_Data-Driven_Charging_Demand_Prediction_at_Public_Charging_Stations_Using_Supervised_Machine_Learning_Regression_Methods (https://www.researchgate.net/publication/343693033_Data-Driven_Charging_Demand_Prediction_at_Public_Charging_Stations_Using_Supervised_Machine_Learning_Regression_Methods)
- In the paper mentioned above, the author has used this historical data along with season, location type and charging fees.
- Since I do not have these information with myself, my results my diverge.
- In here, I will be using Linear Regression, Support Vector Machines, Random Forest and XGBoost machine learning models. I will compare these models.

In [67]:
```
df = pd.concat([caltech_df, jpl_df], axis=0)
```

In [68]:
```
df.shape
```

Out[68]:
```
(65062, 16)
```

In [69]:
```
df.head()
```

Out[69]:

	_id	clusterID	connectionTime	disconnectTime	doneChargingTime	kWhDelivered	sessionID	siteID	spaceID	stationID	
0	5bc90cb9f9af8b0d7fe77cd2	0039	2018-04-25 04:08:04-07:00	2018-04-25 06:20:10-07:00	2018-04-25 06:21:10-07:00	7.932	2_39_78_362_2018-04-25 11:08:04.400812	0002	CA-496	2-39-78-362	Americ
1	5bc90cb9f9af8b0d7fe77cd3	0039	2018-04-25 06:45:10-07:00	2018-04-25 17:56:16-07:00	2018-04-25 09:44:15-07:00	10.013	2_39_95_27_2018-04-25 13:45:09.617470	0002	CA-319	2-39-95-27	Americ
2	5bc90cb9f9af8b0d7fe77cd4	0039	2018-04-25 06:45:50-07:00	2018-04-25 16:04:45-07:00	2018-04-25 07:51:44-07:00	5.257	2_39_79_380_2018-04-25 13:45:49.962001	0002	CA-489	2-39-79-380	Americ
3	5bc90cb9f9af8b0d7fe77cd5	0039	2018-04-25 07:37:06-07:00	2018-04-25 16:55:34-07:00	2018-04-25 09:05:22-07:00	5.177	2_39_79_379_2018-04-25 14:37:06.460772	0002	CA-327	2-39-79-379	Americ
4	5bc90cb9f9af8b0d7fe77cd6	0039	2018-04-25 07:40:34-07:00	2018-04-25 16:03:12-07:00	2018-04-25 10:40:30-07:00	10.119	2_39_79_381_2018-04-25 14:40:33.638896	0002	CA-490	2-39-79-381	Americ

Building a Simple Linear Regression Model

Here we are going to predict the energy delivered based on the session length. Session length is the amount of minutes lapsed between connectionTime and doneChargingTime. Here doneChargingTime is taken because it is the time when the battery became fully charged.

In [70]:
```
simple_df = df.loc[:, ["connectionTime", "doneChargingTime", "kWhDelivered"]]
```

In [71]:
```
d1 = simple_df.copy()
```

In [72]:
```
simple_df.head()
```
Out[72]:

	connectionTime	doneChargingTime	kWhDelivered
0	2018-04-25 04:08:04-07:00	2018-04-25 06:21:10-07:00	7.932
1	2018-04-25 06:45:10-07:00	2018-04-25 09:44:15-07:00	10.013
2	2018-04-25 06:45:50-07:00	2018-04-25 07:51:44-07:00	5.257
3	2018-04-25 07:37:06-07:00	2018-04-25 09:05:22-07:00	5.177
4	2018-04-25 07:40:34-07:00	2018-04-25 10:40:30-07:00	10.119

In [73]:
```
simple_df.shape
```
Out[73]:
```
(65062, 3)
```

In [74]:
```
# Now add a column, "session_length" in the dataframe.
simple_df["session_length"] = (simple_df["doneChargingTime"] - simple_df["connectionTime"])/timedelta(minutes=1)
```

In [75]:
```
simple_df.head()
```
Out[75]:

	connectionTime	doneChargingTime	kWhDelivered	session_length
0	2018-04-25 04:08:04-07:00	2018-04-25 06:21:10-07:00	7.932	133.100000
1	2018-04-25 06:45:10-07:00	2018-04-25 09:44:15-07:00	10.013	179.083333
2	2018-04-25 06:45:50-07:00	2018-04-25 07:51:44-07:00	5.257	65.900000
3	2018-04-25 07:37:06-07:00	2018-04-25 09:05:22-07:00	5.177	88.266667
4	2018-04-25 07:40:34-07:00	2018-04-25 10:40:30-07:00	10.119	179.933333

In [76]:
```
# drop "connectionTime" and "doneChargingTime" columns..
simple_df = simple_df.drop(columns=["connectionTime", "doneChargingTime"])
```

In [77]:
```python
simple_df.head()
```

Out[77]:

	kWhDelivered	session_length
0	7.932	133.100000
1	10.013	179.083333
2	5.257	65.900000
3	5.177	88.266667
4	10.119	179.933333

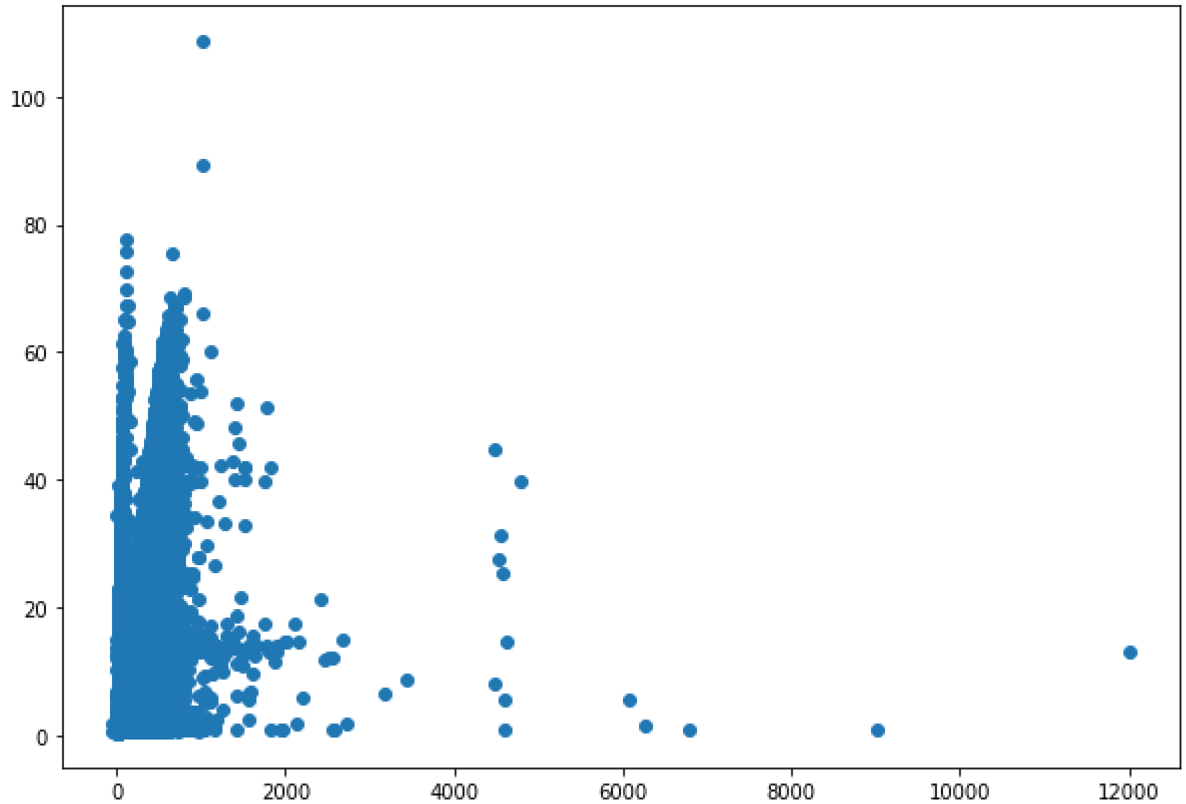
In [78]:
```python
# Check the correlation..
correlation = simple_df.corr()
correlation
```

Out[78]:

	kWhDelivered	session_length
kWhDelivered	1.000000	0.477589
session_length	0.477589	1.000000

In [79]:
```
# Let us plot a scatter plot to furthur understand the dataset..
plt.figure(figsize=(10,7))
plt.scatter(x=simple_df["session_length"],
            y=simple_df["kWhDelivered"])
```

Out[79]:
```
<matplotlib.collections.PathCollection at 0x1c34be3a490>
```



Analysis of the scatter plot :

- There are many EV's which are charged for very small period of time but have been delivered with huge amount of energy. We should indentify these instances from the dataframe.
- There are many EV's which have been charged for more than 2 days but they have been delivered with niegligibly small amount of energy.
- The reason may be : The data that we have been provided has around 4092 missing values in the "doneChargingTime" column. To fill these missing values, we have copied the corresponding values from disconnectTime column. This may be one of the reasons.

In [80]:
```
session_length = list(simple_df["session_length"])
session_length[:10]
session_len_copied = session_length.copy()
```

In [81]:
```
# Let us sort the list in ascending order
session_len_copied.sort()
```

In [82]:
```python
# Analysing to 10 smallest session lengths in the dataframe.
session_len_copied[:50]
```

Out[82]:
```
[-41.36666666666667,
 -40.833333333333336,
 -29.616666666666667,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -1.0,
 -0.9833333333333333,
 -0.9833333333333333,
 -0.9833333333333333,
 -0.9833333333333333,
 -0.9833333333333333,
 -0.9833333333333333,
 -0.9666666666666667,
 -0.9666666666666667,
 -0.9666666666666667,
 -0.9666666666666667,
 -0.95,
 -0.95,
 -0.95,
 -0.95,
 -0.9333333333333333,
 -0.15,
 -0.016666666666666666,
 0.0,
 0.0,
 0.05,
 0.1,
 0.11666666666666667,
 0.43333333333333335,
 0.48333333333333334,
 0.8166666666666667,
 1.0166666666666666,
 1.1666666666666667,
 1.2,
 2.066666666666667,
 2.1333333333333333,
 2.15,
 2.3666666666666667,
 2.55,
```

```
 2.6,
 2.8833333333333333,
 2.9833333333333334,
 3.066666666666667,
 3.6166666666666667,
 3.683333333333333,
 3.716666666666667]
```

How can be session length be negative ?

Done Charging Time should always be greater than the connectionTime. It means there must be some problem with the dataset. Let us see how many session lengths are negative or close to zero.

In [83]:
```
session_length.index(0.8166666666666667)
```

Out[83]:
```
246
```

In [84]:
```
d1.iloc[246]
```

Out[84]:
```
connectionTime      2018-04-30 11:45:38-07:00
doneChargingTime    2018-04-30 11:46:27-07:00
kWhDelivered                         0.586013
Name: 246, dtype: object
```

In [85]:
```
caltech_df.iloc[246]
```

Out[85]:
```
_id                                  5bc915caf9af8b0dad3c0678
clusterID                                                0039
connectionTime                      2018-04-30 11:45:38-07:00
disconnectTime                      2018-04-30 16:22:23-07:00
doneChargingTime                    2018-04-30 11:46:27-07:00
kWhDelivered                                         0.586013
sessionID           2_39_95_444_2018-04-30 18:45:38.019636
siteID                                                   0002
spaceID                                                CA-497
stationID                                          2-39-95-444
timezone                                  America/Los_Angeles
userID                                              unclaimed
userInputs                                               None
session_duration                                       276.75
Day                                                   weekDay
connectionDate                                     2018-04-30
Name: 246, dtype: object
```

Here EV at index number 246 has been charged for around 1 minute but has consumed 0.586 kWh of energy. It seems there is some problem here. The Ev was connected at 11:45 AM and disconnected at 4:22 PM but its battery became fully charged at 11:46 AM.

In [86]:
```
session_length.index(-1.0)
```

Out[86]:
```
494
```

In [87]:
```
d1.iloc[494]
```

Out[87]:
```
connectionTime      2018-05-04 12:23:52-07:00
doneChargingTime    2018-05-04 12:22:52-07:00
kWhDelivered                         0.912297
Name: 494, dtype: object
```

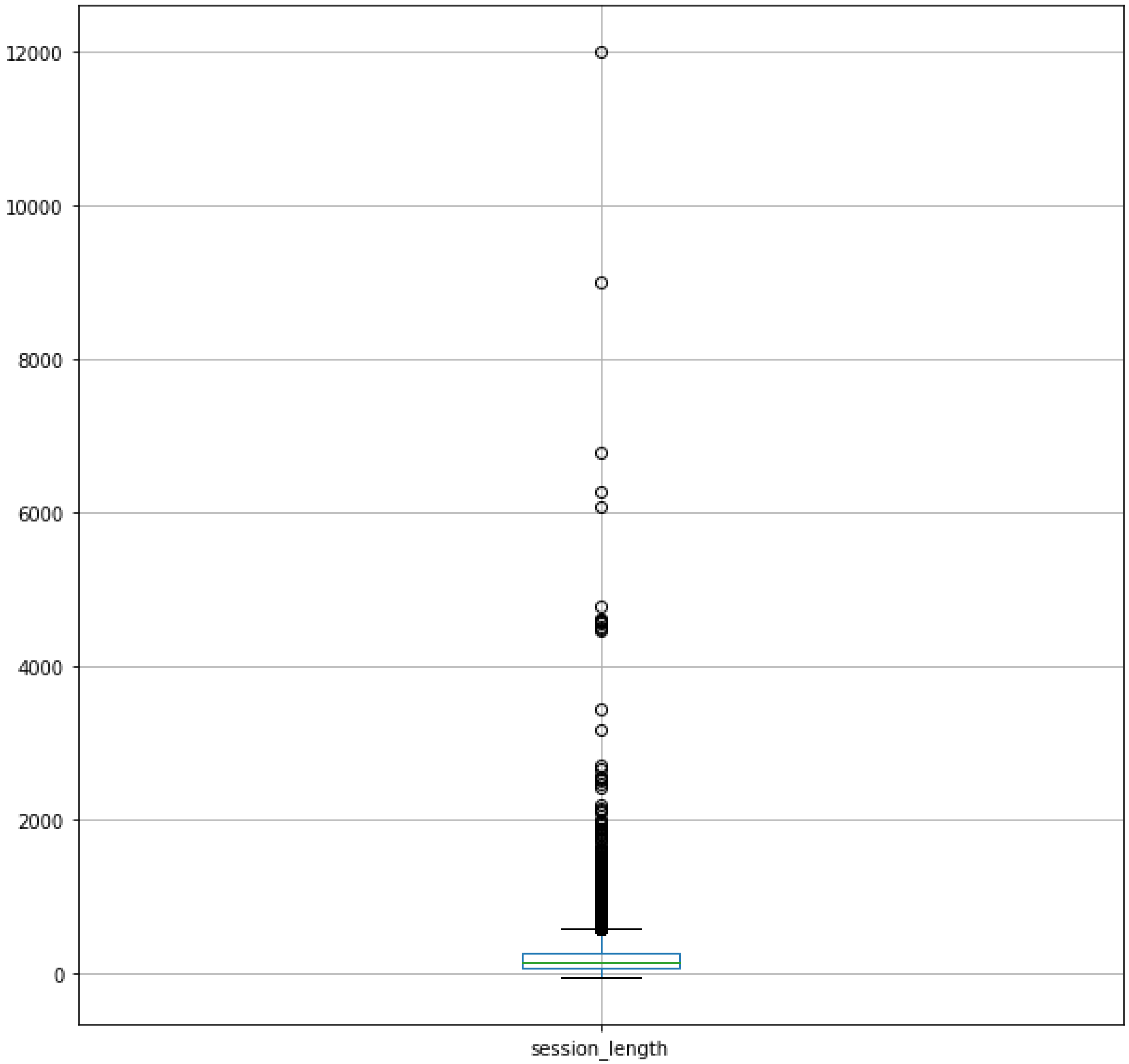
In [88]:
```
caltech_df.iloc[494]
```

Out[88]:
```
_id                                 5bc91740f9af8b0dc677b862
clusterID                                               0039
connectionTime                     2018-05-04 12:23:52-07:00
disconnectTime                     2018-05-04 17:04:15-07:00
doneChargingTime                   2018-05-04 12:22:52-07:00
kWhDelivered                                        0.912297
sessionID           2_39_78_367_2018-05-04 19:23:51.897392
siteID                                                  0002
spaceID                                               CA-494
stationID                                        2-39-78-367
timezone                                 America/Los_Angeles
userID                                             unclaimed
userInputs                                              None
session_duration                                  280.383333
Day                                                  weekDay
connectionDate                                    2018-05-04
Name: 494, dtype: object
```

Here also the EV was connected at 12:23 PM and disconnected at 5:00 PM but the EV was fully charged at 12:22 PM, which seems a bit odd. So let us find these outliers and remove them from our dataframe.

In [89]:
```
plt.figure(figsize=(10,10))
simple_df[["session_length"]].boxplot()
```

Out[89]: <AxesSubplot:>



In [90]:
```
for x in ['session_length']:
    q75,q25 = np.percentile(simple_df.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)

    simple_df.loc[simple_df[x] < min,x] = np.nan
    simple_df.loc[simple_df[x] > max,x] = np.nan
```

In [91]:
```
simple_df["session_length"].isnull().sum()
```

Out[91]:
```
1812
```

Hence there are 1812 outliers in the session length column of the dataframe. We have to remove these rows.

In [92]:
```
simple_df = simple_df.dropna()
```

In [93]:
```
simple_df["session_length"].isnull().sum()
```

Out[93]:
```
0
```

In [94]:
```
simple_df.shape # From 65062, we have been left with 63250 instances only.
```

Out[94]:
```
(63250, 2)
```

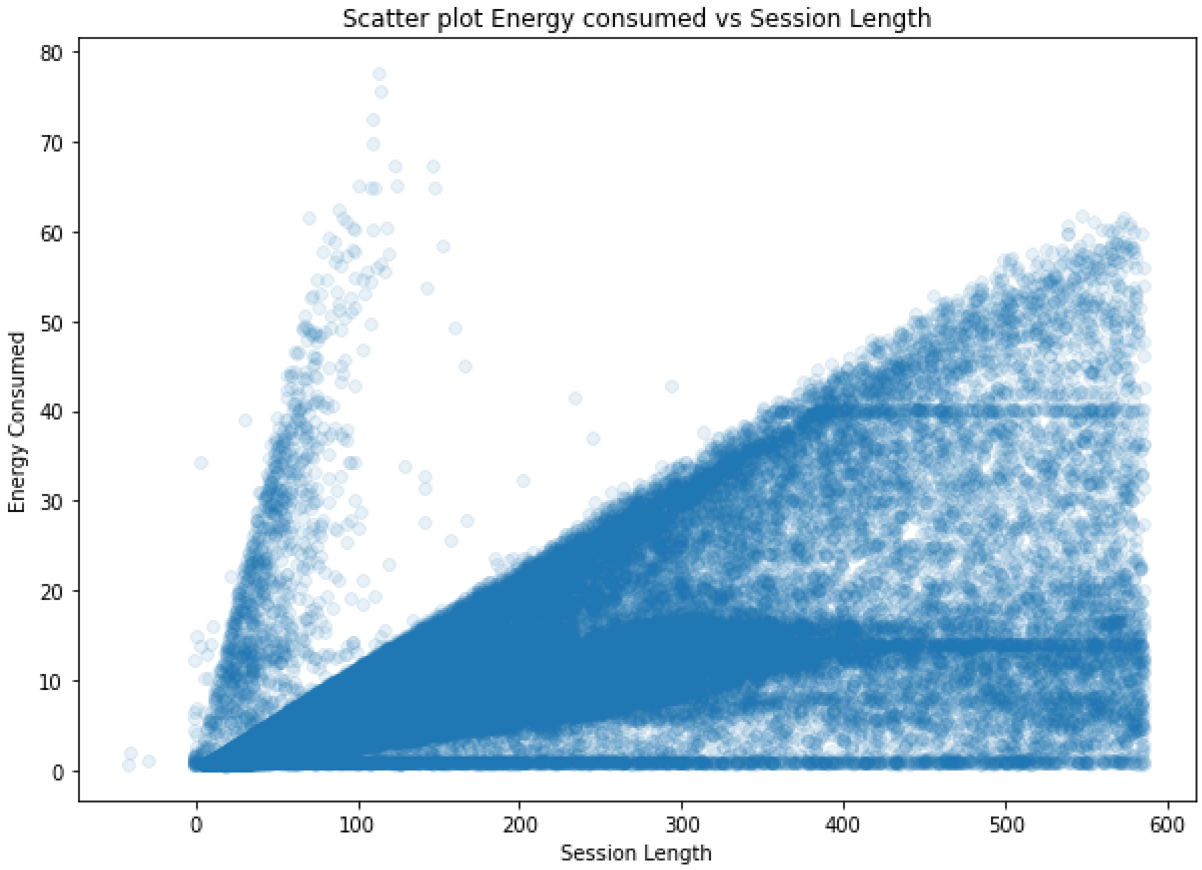
In [95]:
```
# Now let us find the correlation again.
correlation = simple_df.corr()
correlation
```

Out[95]:

	kWhDelivered	session_length
kWhDelivered	1.000000	0.594835
session_length	0.594835	1.000000

The correlation between kWhDelivered and session_length columns was around 48% before the removal of outliers has been improved to 60% after the removal of the outliers. This increment is significant.

In [96]:
```
plt.figure(figsize=(10,7))
plt.scatter(x=simple_df["session_length"],
            y=simple_df["kWhDelivered"],
            alpha=0.1)
plt.title("Scatter plot Energy consumed vs Session Length")
plt.xlabel("Session Length")
plt.ylabel("Energy Consumed")
plt.show()
```



- The figure shows some horizontal lines at 40 kWh, 15 kWh and closer to 0 kWh. It means there must be some kind of capping at these values.

Splitting the dataset into a train and test set

In [97]:
```
simple_df.shape
```

Out[97]: (63250, 2)

In [98]:
```
# Shuffle the dataframe
simple_df = simple_df.sample(frac=1, random_state=42)
```

In [99]:
```
X = simple_df[["session_length"]]
y = simple_df[["kWhDelivered"]]
```

In [100]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [101]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[101]: ((50600, 1), (12650, 1), (50600, 1), (12650, 1))

In [102]:
```
simple_df.head()
```

Out[102]:

	kWhDelivered	session_length
27383	0.805	30.583333
7402	32.392	365.216667
7273	2.431	52.600000
14558	10.755	448.150000
282	4.560	104.683333

Build a function to calculate the model performance

In [103]:
```
def calculate_performance(y_test, y_pred):
    MAE = np.round(mean_absolute_error(y_test, y_pred),2)
    RMSE = np.round(np.sqrt(mean_squared_error(y_test, y_pred)),2)
    r_sq = np.round(r2_score(y_test, y_pred),2)

    performance_dic = {"MAE": MAE, "RMSE" : RMSE, "r2_score" : r_sq}
    return performance_dic
```

Model Building

Model 1 : Linear Regression

In [104]:
```
model_1_lr = LinearRegression()
```

In [105]:
```
# Fit the training data into the model..
model_1_lr.fit(X_train, y_train)
```

Out[105]:
```
LinearRegression()
```

In [106]:
```
# Our model has been trained. Let us predict on test datasets.
y_pred = model_1_lr.predict(X_test)
```

In [107]:
```
model_1_performance = calculate_performance(y_test, y_pred)
```

In [108]:
```
model_1_performance
```

Out[108]:
```
{'MAE': 5.45, 'RMSE': 8.03, 'r2_score': 0.36}
```

Model 2 : Random Forest Regressor

In [109]:
```
model_2_rf = RandomForestRegressor()
```

In [110]:
```
model_2_rf.fit(X_train, y_train)
```

Out[110]:
```
RandomForestRegressor()
```

In [111]:
```
y_pred_rf = model_2_rf.predict(X_test)
```

In [112]:
```
model_2_performance = calculate_performance(y_test, y_pred_rf)
model_2_performance
```

Out[112]:
```
{'MAE': 6.08, 'RMSE': 9.41, 'r2_score': 0.12}
```

In [113]:
```
y_test[:10], y_pred_rf[:10]
```

Out[113]:
```
(       kWhDelivered
 6198         10.874
 8626          9.528
 25605         9.644
 17615         5.844
 16274         7.844
 22046        18.139
 26927         5.375
 25517        11.265
 31114        10.959
 18628        35.026,
 array([ 8.40012848,  8.47118059, 14.6168936 , 17.76029866,  6.74843883,
        19.71022256,  8.29801626,  8.73977917, 16.95342   , 32.573825  ]))
```

Using Cross validation to train the Random Forest Model

In [114]:
```
scores = cross_val_score(model_2_rf, X, y, scoring="neg_mean_squared_error", cv=10)
```

In [115]:
```
rmse_scores = np.sqrt(-scores)
```

In [116]:
```
rmse = np.mean(rmse_scores)
```

In [117]:
```
rmse
```

Out[117]:
```
9.24284081271295
```

Model 3 : Support Vector Machine

In [118]:
```
model_3_svr = SVR()
```

In [119]:
```
model_3_svr.fit(X_train, y_train)
```

Out[119]:
```
SVR()
```

In [120]:
```
y_pred_svr = model_3_svr.predict(X_test)
```

In [121]:
```
model_3_performance = calculate_performance(y_test, y_pred_svr)
model_3_performance
```
Out[121]:
```
{'MAE': 5.12, 'RMSE': 8.4, 'r2_score': 0.3}
```

Model 4 : XGBoost

In [122]:
```
from xgboost import XGBRegressor
```
In [123]:
```
model_4_xgb = XGBRegressor()
```
In [124]:
```
model_4_xgb.fit(X_train, y_train)
```
Out[124]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=2,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```
In [125]:
```
y_pred_xgb = model_4_xgb.predict(X_test)
```
In [126]:
```
model_4_performance = calculate_performance(y_test, y_pred_xgb)
model_4_performance
```
Out[126]:
```
{'MAE': 5.41, 'RMSE': 8.15, 'r2_score': 0.34}
```

Comparing the results of all 4 models

In [127]:
```
results_dic = {"Linear Regression" : model_1_performance,
               "Random Forest" : model_2_performance,
               "Support Vector Machines" : model_3_performance,
               "XGBoost" : model_4_performance}
```

In [128]:
```
results_df = pd.DataFrame(results_dic)
results_df
```

Out[128]:

	Linear Regression	Random Forest	Support Vector Machines	XGBoost
MAE	5.45	6.08	5.12	5.41
RMSE	8.03	9.41	8.40	8.15
r2_score	0.36	0.12	0.30	0.34