Data Preprocessing

In [200]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [201]:

```
# Importing the dataset
df = pd.read_csv('RL_SR.csv', sep=';')
```

In [202]:

```
df
```

Out[202]:

	ActualPower	Max Capacity	Location 1	Location 2	Location 3	Location 4
0	0.004	45.00	3.9	2.907	0.237	2.281
1	0.423	45.00	4.2	3.069	0.254	2.477
2	0.805	45.00	4.0	4.226	0.249	3.577
3	1.985	45.00	4.0	4.223	0.317	3.685
4	4.492	45.00	3.3	4.107	0.288	3.619
...	...	...	...	...	...	...
289	1.406	43.75	4.0	5.600	5.733	6.415
290	1.200	43.75	3.2	1.722	2.591	1.489
291	2.147	43.75	3.6	1.767	2.469	1.500
292	1.234	43.75	3.9	1.781	2.387	1.505
293	3.378	43.75	4.3	1.789	2.311	1.520

294 rows × 6 columns

In [203]:

```
df.describe()
# Критических выбросов не наблюдается
```

Out[203]:

	ActualPower	Max Capacity	Location 1	Location 2	Location 3	Location 4
count	293.000000	292.000000	294.000000	293.00000	294.000000	293.000000
mean	16.643478	43.904966	7.467347	8.34471	5.878480	8.623092
std	9.820152	1.829667	2.413660	1.73226	1.921458	1.873867
min	0.004000	36.000000	0.000000	1.72200	0.137000	1.489000
25%	9.507000	43.750000	6.100000	7.61300	5.328250	8.093000
50%	14.689000	43.750000	7.400000	8.48100	6.127000	8.771000
75%	23.340000	45.000000	8.800000	9.37300	6.953250	9.888000
max	37.219000	45.000000	14.400000	11.47800	9.068000	11.962000

In [204]:

```
# mean()-3*std
# Let's check how much the data are spread out from the mean.
mean_ActualPower = np.mean(df['ActualPower'], axis=0)
sd_ActualPower = np.std(df['ActualPower'], axis=0)

mean_MaxCapacity = np.mean(df['Max Capacity'], axis=0)
sd_MaxCapacity = np.std(df['Max Capacity'], axis=0)

mean_Location1 = np.mean(df['Location 1'], axis=0)
sd_Location1 = np.std(df['Location 1'], axis=0)

mean_Location2 = np.mean(df['Location 2'], axis=0)
sd_Location2 = np.std(df['Location 2'], axis=0)

mean_Location3 = np.mean(df['Location 3'], axis=0)
sd_Location3 = np.std(df['Location 3'], axis=0)

mean_Location4 = np.mean(df['Location 4'], axis=0)
sd_Location4 = np.std(df['Location 4'], axis=0)

counter_actual_power = 0
counter_maxcapacity = 0
counter_loc1 = 0
counter_loc2 = 0
counter_loc3 = 0
counter_loc4 = 0

for actual_power, maxcapacity, loc1, loc2, loc3, loc4 in zip(df['ActualPower'], df['Max
Capacity'], df['Location 1'], df['Location 2'], df['Location 3'], df['Location 4']):
    if not mean_ActualPower - 3*sd_ActualPower <= actual_power <= mean_ActualPower + 3*
sd_ActualPower:
        counter_actual_power += 1
    if not mean_MaxCapacity - 3*sd_MaxCapacity <= maxcapacity <= mean_MaxCapacity + 3*s
d_MaxCapacity:
        counter_maxcapacity += 1
    if not mean_Location1 - 3*sd_Location1 <= counter_loc1 <= mean_Location1 + 3*sd_Loc
ation1:
        counter_loc1 += 1
    if not mean_Location2 - 3*sd_Location2 <= counter_loc2 <= mean_Location2 + 3*sd_Loc
ation2:
        counter_loc2 += 1
    if not mean_Location3 - 3*sd_Location3 <= counter_loc3 <= mean_Location3 + 3*sd_Loc
ation3:
        counter_loc3 += 1
    if not mean_Location4 - 3*sd_Location4 <= counter_loc4 <= mean_Location4 + 3*sd_Loc
ation4:
        counter_loc4 += 1

counter_dicts = {'counter_actual_power': counter_actual_power,
                'counter_maxcapacity': counter_maxcapacity,
                'counter_loc1': counter_loc1,
                'counter_loc2': counter_loc2,
                'counter_loc3': counter_loc3,
                'counter_loc4': counter_loc4}
print(counter_dicts)
```

```
{'counter_actual_power': 1, 'counter_maxcapacity': 17, 'counter_loc1': 1,
'counter_loc2': 4, 'counter_loc3': 1, 'counter_loc4': 4}
```

In [205]:

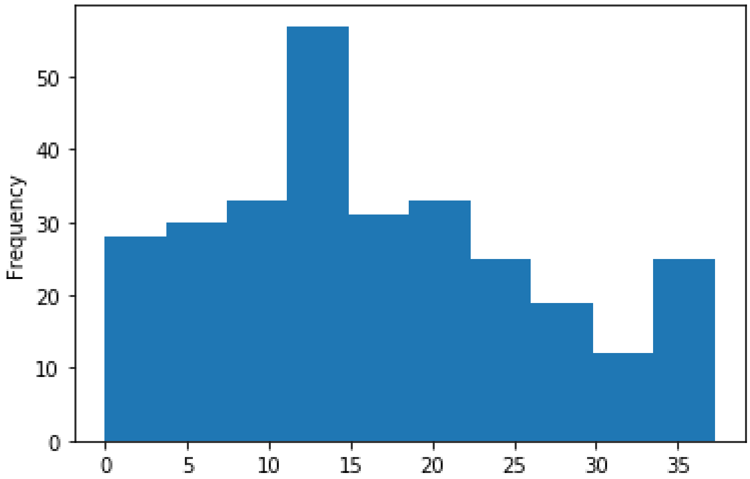
```
# Outliers
actual_power = []
for ap in df['ActualPower']:
    if ap > df['ActualPower'].mean() + 3 * df['ActualPower'].std():
        ap = df['ActualPower'].mean() + 3*df['ActualPower'].std()
    elif ap < df['ActualPower'].mean() - 3 * df['ActualPower'].std():
        ap = df['ActualPower'].mean() - 3*df['ActualPower'].std()
    actual_power.append(ap)
df['ActualPower']  = actual_power

maxcapacity = []
for m in df['Max Capacity']:
    if m > df['Max Capacity'].mean() + 3 * df['Max Capacity'].std():
        m = df['Max Capacity'].mean() + 3*df['Max Capacity'].std()
    elif m < df['Max Capacity'].mean() - 3 * df['Max Capacity'].std():
        m = df['Max Capacity'].mean() - 3*df['Max Capacity'].std()
    maxcapacity.append(m)
df['Max Capacity']  = maxcapacity

loc1 = []
for loc in df['Location 1']:
    if loc > df['Location 1'].mean() + 3 * df['Location 1'].std():
        loc = df['Location 1'].mean() + 3*df['Location 1'].std()
    elif loc < df['Location 1'].mean() - 3 * df['Location 1'].std():
        loc = df['Location 1'].mean() - 3*df['Location 1'].std()
    loc1.append(loc)
df['Location 1'] = loc1

loc2 = []
for loc in df['Location 2']:
    if loc > df['Location 2'].mean() + 3 * df['Location 2'].std():
        loc = df['Location 2'].mean() + 3*df['Location 2'].std()
    elif loc < df['Location 2'].mean() - 3 * df['Location 2'].std():
        loc = df['Location 2'].mean() - 3*df['Location 2'].std()
    loc2.append(loc)
df['Location 2'] = loc2

loc3 = []
for loc in df['Location 3']:
    if loc > df['Location 3'].mean() + 3 * df['Location 3'].std():
        loc = df['Location 3'].mean() + 3*df['Location 3'].std()
    elif loc < df['Location 3'].mean() - 3 * df['Location 3'].std():
        loc = df['Location 3'].mean() - 3*df['Location 3'].std()
    loc3.append(loc)
df['Location 3'] = loc3

loc4 = []
for loc in df['Location 4']:
    if loc > df['Location 4'].mean() + 3 * df['Location 4'].std():
        loc = df['Location 4'].mean() + 3*df['Location 4'].std()
    elif loc < df['Location 4'].mean() - 3 * df['Location 4'].std():
        loc = df['Location 4'].mean() - 3*df['Location 4'].std()
    loc4.append(loc)
df['Location 4'] = loc4
```

In [206]:

```
# ActualPower distribution
df['ActualPower'].plot(kind = 'hist')
```

Out[206]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c4a55d4580>

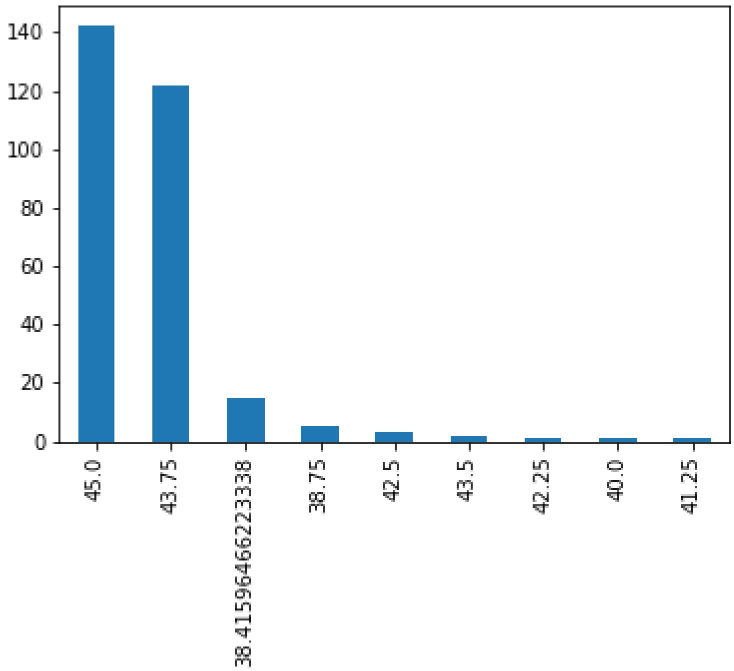


In [207]:

```
# Max Capacity distribution
distribution = df['Max Capacity'].value_counts()
distribution.plot(kind='bar')
```

Out[207]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c4a55beeb0>

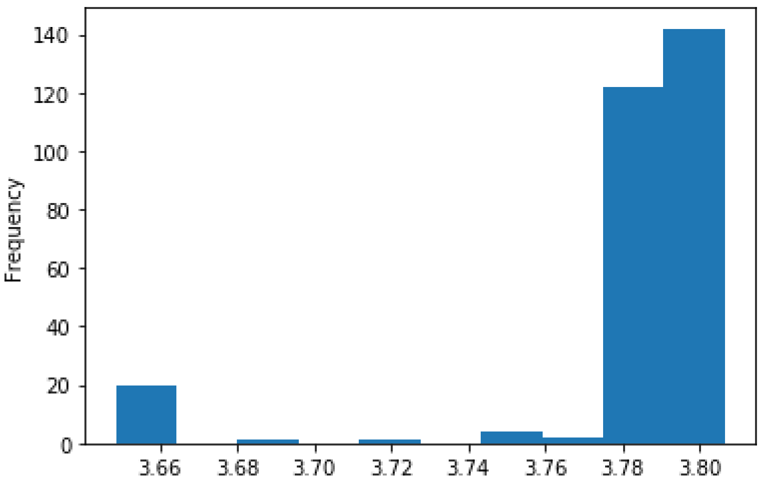


In [208]:

```
# Max Capacity distribution log
distribution = np.log(df['Max Capacity'])
distribution.plot(kind = 'hist')
```

Out[208]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a56e5ee0>
```

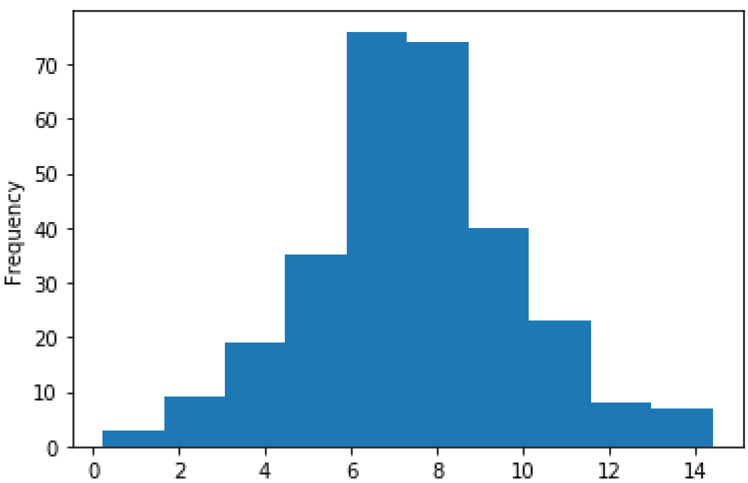


In [209]:

```
# Location 1 distribution
df['Location 1'].plot(kind = 'hist')
```

Out[209]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a55479a0>
```

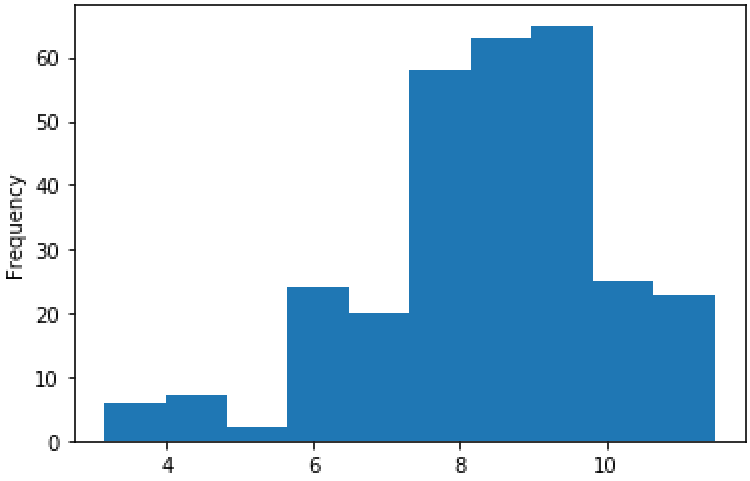


In [210]:

```
# Location 2 distribution
df['Location 2'].plot(kind = 'hist')
```

Out[210]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a5475280>
```

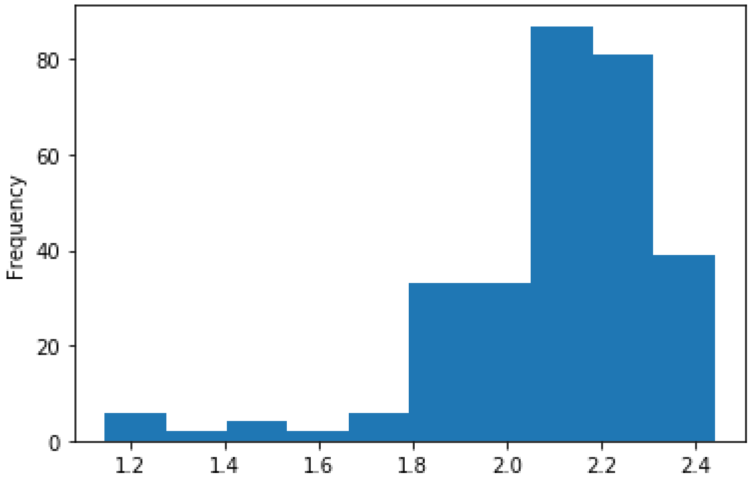


In [211]:

```
# Location 2 distribution log
distribution = np.log(df['Location 2'])
distribution.plot(kind = 'hist')
```

Out[211]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a4f91b80>
```

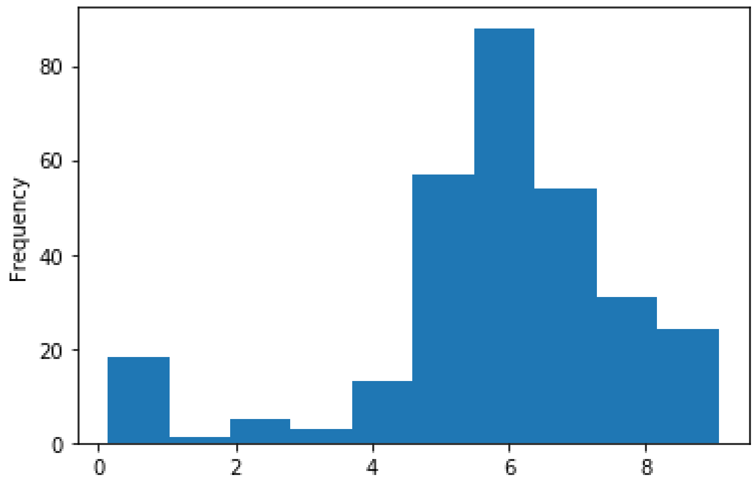


In [212]:

```
# Location 3 distribution
df['Location 3'].plot(kind = 'hist')
```

Out[212]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a3e410d0>
```

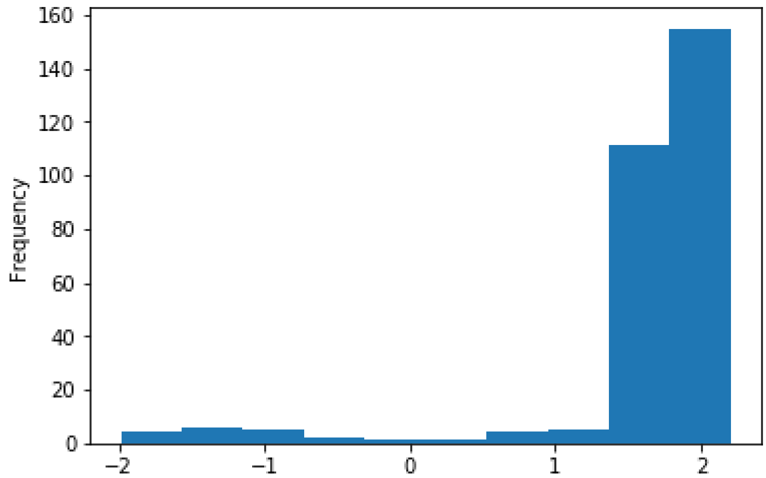


In [213]:

```
# Location 3 distribution log
distribution = np.log(df['Location 3'])
distribution.plot(kind = 'hist')
```

Out[213]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a3a38790>
```

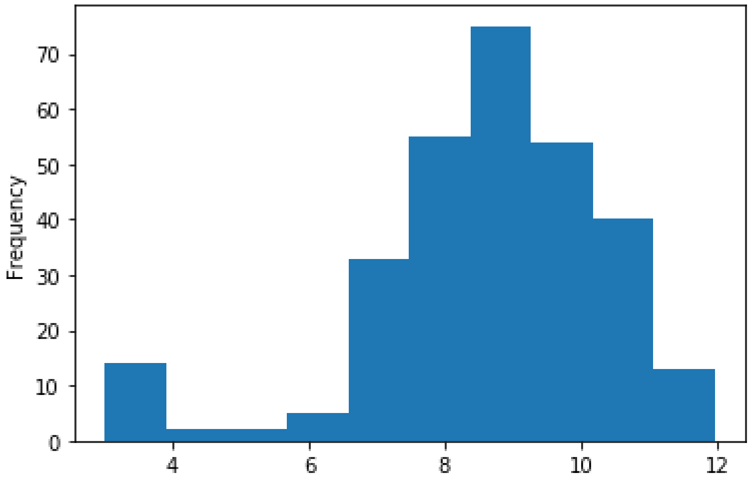


In [214]:

```
# Location 4 distribution
df['Location 4'].plot(kind = 'hist')
```

Out[214]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a564ad30>
```

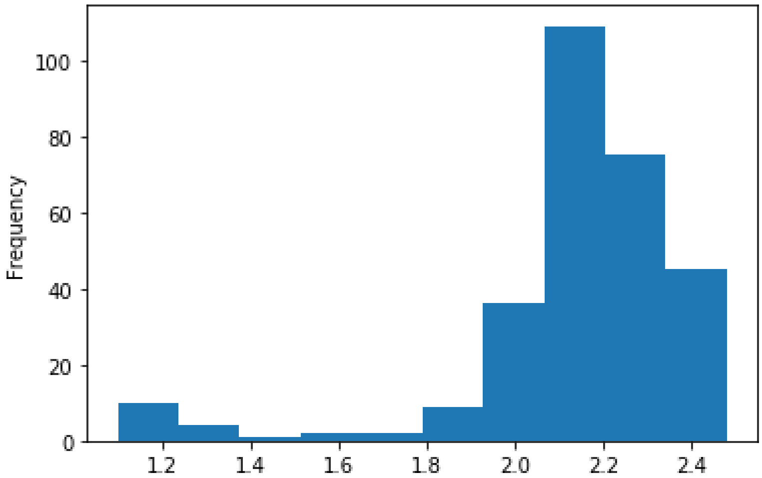


In [215]:

```
# Location 4 distribution log
distribution = np.log(df['Location 4'])
distribution.plot(kind = 'hist')
```

Out[215]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a5110df0>
```



In [216]:

```
df.isnull().sum()
# Таким образом мы имеем пропущенные значения в таких колонках:
```

Out[216]:

```
ActualPower     1
Max Capacity    2
Location 1      0
Location 2      1
Location 3      0
Location 4      1
dtype: int64
```

In [217]:

```
# Taking care of missing data
# https://scikit-learn.org/
from sklearn.impute import SimpleImputer
#numeric
df[['ActualPower']] = SimpleImputer(missing_values=np.nan, strategy='mean').fit_transfo
rm(df[['ActualPower']]).round()
df[['Max Capacity']] = SimpleImputer(missing_values=np.nan, strategy='mean').fit_transf
orm(df[['Max Capacity']]).round()
df[['Location 2']] = SimpleImputer(missing_values=np.nan, strategy='mean').fit_transfor
m(df[['Location 2']]).round()
df[['Location 4']] = SimpleImputer(missing_values=np.nan, strategy='mean').fit_transfor
m(df[['Location 4']]).round()
```

In [218]:

```
df_log = pd.DataFrame()
df_log['ActualPower']=df['ActualPower']
df_log['Max Capacity']=np.log(df['Max Capacity'])
df_log['Location 1']=df['Location 1']
df_log['Location 2']=np.log(df['Location 2'])
df_log['Location 3']=np.log(df['Location 3'])
df_log['Location 4']=np.log(df['Location 4'])
```

Linear Regression

In [219]:

```
df_log
```

Out[219]:

	ActualPower	Max Capacity	Location 1	Location 2	Location 3	Location 4
0	0.0	3.806662	3.9	1.098612	-1.439695	1.098612
1	0.0	3.806662	4.2	1.098612	-1.370421	1.098612
2	1.0	3.806662	4.0	1.386294	-1.390302	1.386294
3	2.0	3.806662	4.0	1.386294	-1.148854	1.386294
4	4.0	3.806662	3.3	1.386294	-1.244795	1.386294
...	...	...	...	...	...	...
289	1.0	3.784190	4.0	1.791759	1.746239	1.791759
290	1.0	3.784190	3.2	1.098612	0.952044	1.098612
291	2.0	3.784190	3.6	1.098612	0.903813	1.098612
292	1.0	3.784190	3.9	1.098612	0.870037	1.098612
293	3.0	3.784190	4.3	1.098612	0.837680	1.098612

294 rows × 6 columns

In [220]:

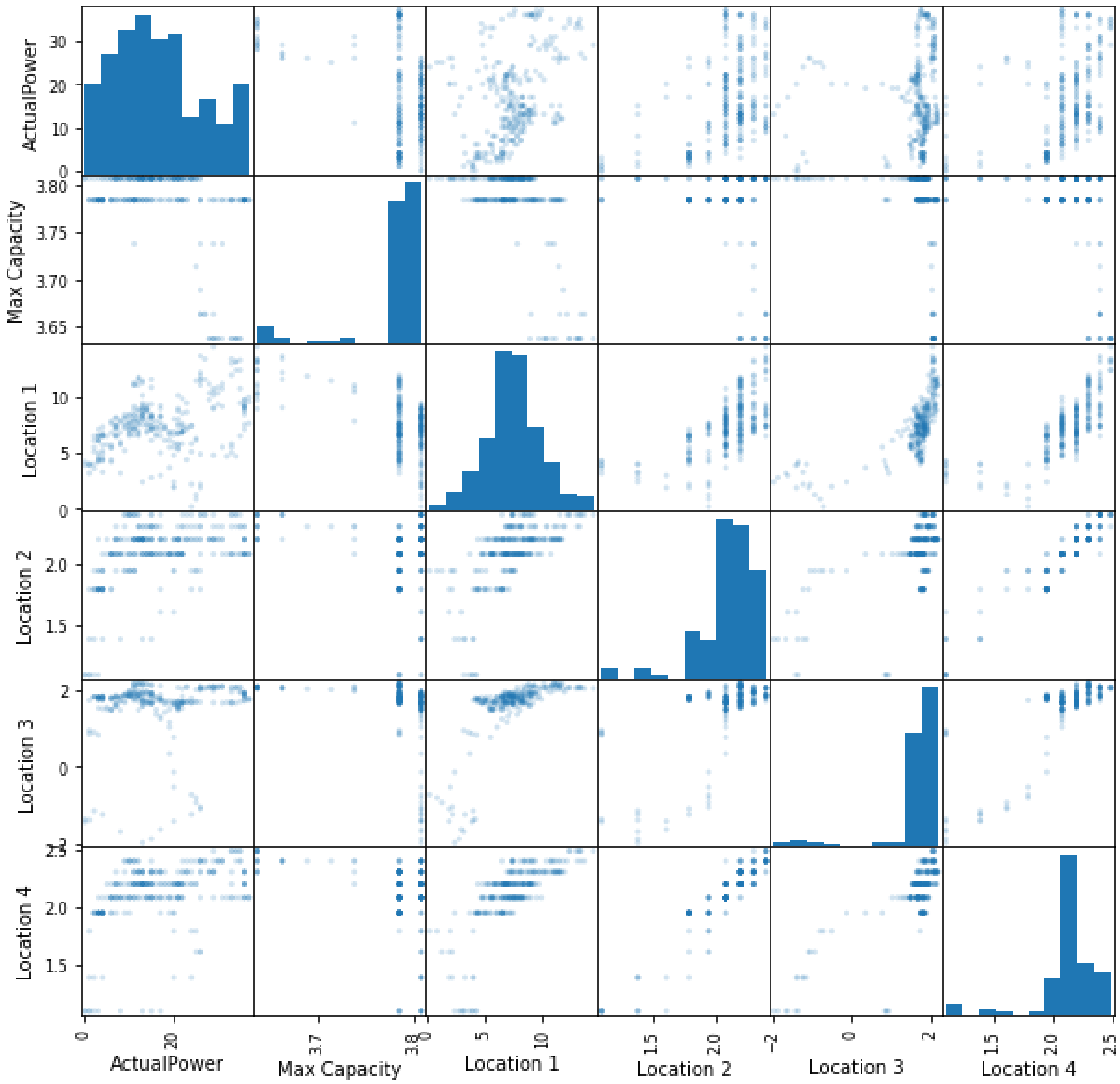
```
# Cheking correlations
df_log.corr()
```

Out[220]:

	ActualPower	Max Capacity	Location 1	Location 2	Location 3	Location 4
ActualPower	1.000000	-0.407536	0.379332	0.433053	0.115857	0.393509
Max Capacity	-0.407536	1.000000	-0.536676	-0.199666	-0.255722	-0.301019
Location 1	0.379332	-0.536676	1.000000	0.625311	0.656575	0.721600
Location 2	0.433053	-0.199666	0.625311	1.000000	0.626065	0.932485
Location 3	0.115857	-0.255722	0.656575	0.626065	1.000000	0.794625
Location 4	0.393509	-0.301019	0.721600	0.932485	0.794625	1.000000

In [221]:

```
from pandas.plotting import scatter_matrix
scatter_matrix(df_log, alpha=0.2, figsize=(10, 10))
plt.show()
```



In [222]:

```
# Splitting the dataset into the Training set and Test set
X = df_log.iloc[:, 1:6].values
y = df_log.iloc[:, 0].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0
)
```

In [223]:

```
# Fitting Simple Linear Regression to the Training set (ActualPower)
from sklearn.linear_model import LinearRegression
sr = LinearRegression().fit(X_train[:, 2:3], y_train)
```

In [224]:

```
# Getting parameters
sr.coef_, sr.intercept_
```

Out[224]:

```
(array([17.91103715]), -21.049657078580022)
```

In [225]:

```
# Predicting the Test set results
y_pred = sr.predict(X_test[:, 2:3])
```

In [226]:

```
# Coefficient of determination R^2
sr.score(X_train[:, 2:3], y_train), sr.score(X_test[:, 2:3], y_test)
```

Out[226]:

```
(0.1908749365272442, 0.17233498442617246)
```

In [227]:

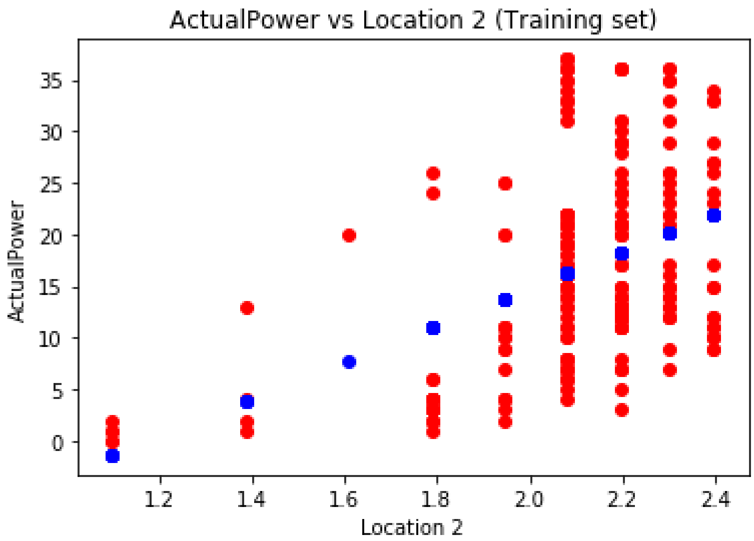
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, sr.predict(X_train[:, 2:3])), mean_squared_error(y_test, y_
pred)
```

Out[227]:

```
(77.54786471167098, 80.32036031055034)
```

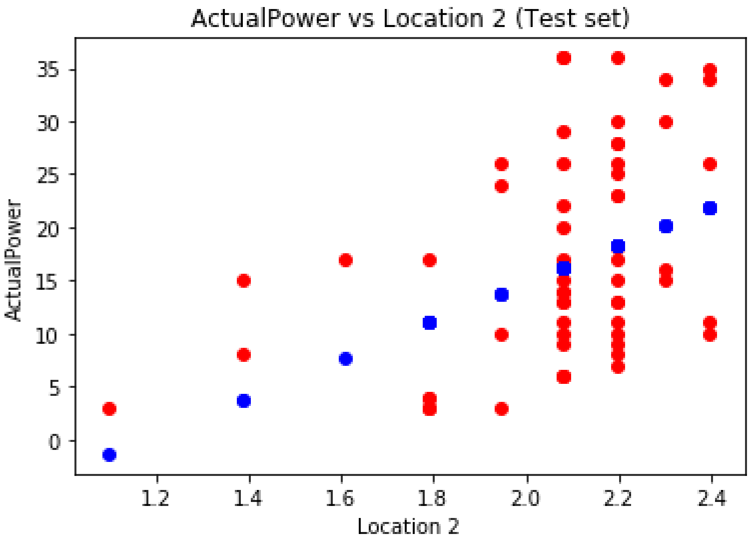
In [228]:

```
# Visualising the Training set results
plt.scatter(X_train[:,2], y_train, color = 'red')
plt.plot(X_train[:,2], sr.predict(X_train[:, 2:3]), 'bo')
plt.title('ActualPower vs Location 2 (Training set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [229]:

```
# Visualising the Test set results
plt.scatter(X_test[:,2], y_test, color = 'red')
plt.plot(X_test[:,2], sr.predict(X_test[:, 2:3]), 'bo')
plt.title('ActualPower vs Location 2 (Test set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [230]:

```
# Fitting Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
mr = LinearRegression().fit(X_train, y_train)
```

In [231]:

```
# Getting parameters
mr.coef_, mr.intercept_
```

Out[231]:

```
(array([-80.32771757,   0.43800275,  14.7619549 ,  -5.71477801,
          9.2965157 ]),
 276.04715900565003)
```

In [232]:

```
# Predicting the Test set results
y_pred = mr.predict(X_test)
```

In [233]:

```
# Coefficient of determination R^2
mr.score(X_train, y_train), mr.score(X_test, y_test)
```

Out[233]:

```
(0.3510663176818859, 0.44470012975574835)
```

In [234]:

```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, mr.predict(X_train)), mean_squared_error(y_test, y_pred)
```

Out[234]:

```
(62.194861678445626, 53.88881349237319)
```

In [235]:

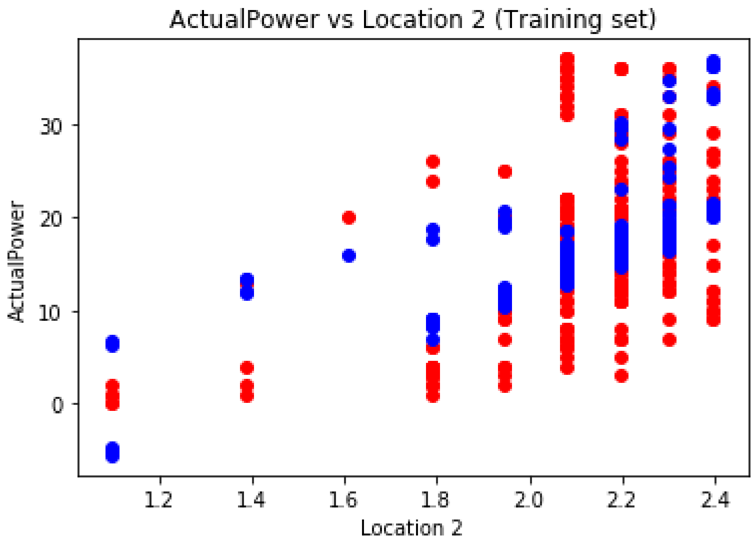
```
# !pip install statsmodels
# p-values
import statsmodels.api as sm
X = sm.add_constant(X_train)
mr1 = sm.OLS(y_train, X).fit()
mr1.pvalues
#mr1.summary()
```

Out[235]:

```
array([5.70052315e-05, 7.64209939e-06, 2.47648080e-01, 4.69894962e-02,
       1.94119385e-05, 2.94837682e-01])
```

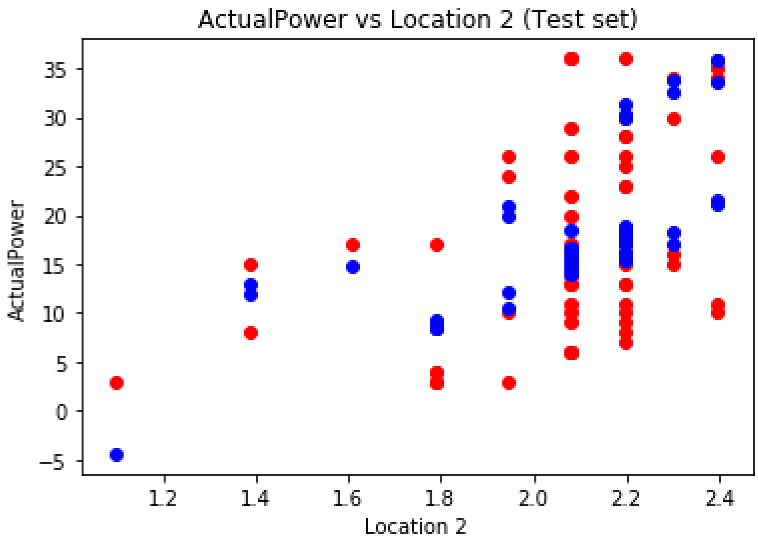
In [236]:

```
# Visualising the Training set results
plt.scatter(X_train[:,2], y_train, color = 'red')
plt.plot(X_train[:,2], mr.predict(X_train), 'bo')
plt.title('ActualPower vs Location 2 (Training set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [237]:

```
# Visualising the Test set results
plt.scatter(X_test[:,2], y_test, color = 'red')
plt.plot(X_test[:,2], mr.predict(X_test), 'bo')
plt.title('ActualPower vs Location 2 (Test set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [238]:

```
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
X_train_p = PolynomialFeatures().fit_transform(X_train[:, 2:3])
X_test_p = PolynomialFeatures().fit_transform(X_test[:, 2:3])
pr = LinearRegression().fit(X_train_p[:,1:], y_train)
```

In [239]:

```
# Getting parameters
pr.coef_, pr.intercept_
```

Out[239]:

```
(array([18.1034046 , -0.05165562]), -21.22284326775601)
```

In [240]:

```
# Predicting the Test set results
y_pred = pr.predict(X_test_p[:,1:])
```

In [241]:

```
# Coefficient of determination R^2
pr.score(X_train_p[:,1:], y_train), pr.score(X_test_p[:,1:], y_test)
```

Out[241]:

```
(0.19087526134588262, 0.1721999186006038)
```

In [242]:

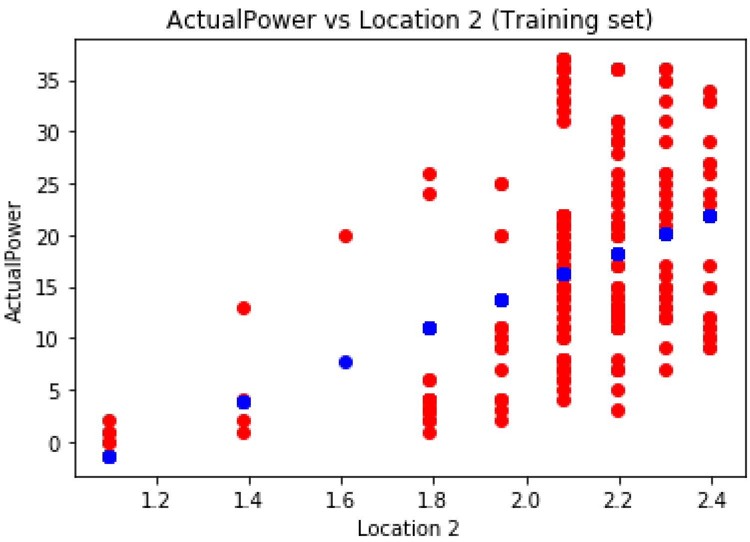
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, pr.predict(X_train_p[:,1:])), mean_squared_error(y_test, y_
pred)
```

Out[242]:

```
(77.54783358052332, 80.33346770976522)
```

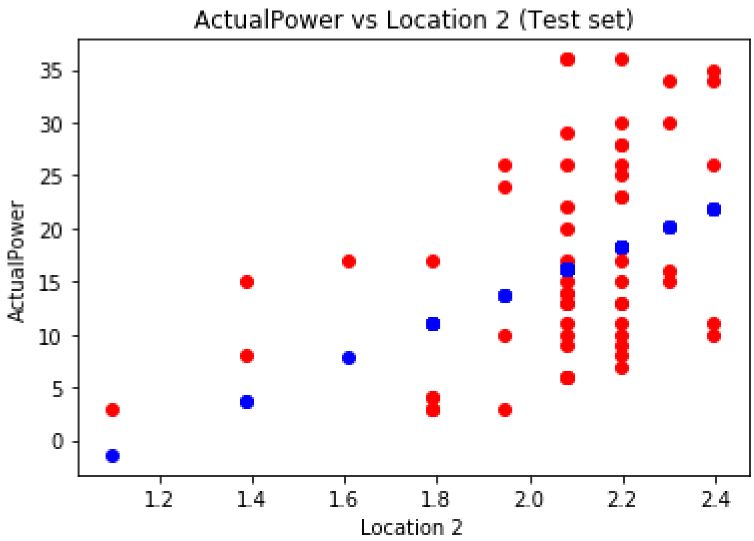
In [243]:

```
# Visualising the Training set results
plt.scatter(X_train[:,2], y_train, color = 'red')
plt.plot(X_train[:,2], pr.predict(X_train_p[:,1:]), 'bo')
plt.title('ActualPower vs Location 2 (Training set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [244]:

```
# Visualising the Test set results
plt.scatter(X_test[:,2], y_test, color = 'red')
plt.plot(X_test[:,2], pr.predict(X_test_p[:,1:]), 'bo')
plt.title('ActualPower vs Location 2 (Test set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [245]:

```
# Backward Elimination with p-values
import statsmodels.api as sm
def backwardElimination(x, sl):
    numVars = len(x[0])
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    x = np.delete(x, j, 1)
    regressor_OLS.summary()
    return x

SL = 0.05
X_opt = X_train[:, [0, 1, 2, 3, 4]]
y = y_train
X_Modeled = backwardElimination(X_opt, SL)
```

In [246]:

```
X_Modeled
```

Out[246]:

```
array([[ 3.80666249,  7.6       ,  1.83848401,  2.19722458],
       [ 3.80666249,  7.3       ,  1.86810304,  2.39789527],
       [ 3.80666249,  6.1       ,  1.03673688,  2.07944154],
       [ 3.80666249,  6.8       ,  1.95302762,  2.19722458],
       [ 3.80666249,  7.8       ,  1.82358108,  2.07944154],
       [ 3.78418963,  9.3       ,  2.10132534,  2.39789527],
       [ 3.78418963,  4.7       ,  1.67109729,  2.07944154],
       [ 3.80666249,  6.9       ,  1.53772706,  2.07944154],
       [ 3.78418963,  6.5       ,  1.71054941,  2.07944154],
       [ 3.78418963,  4.        ,  1.74623895,  1.79175947],
       [ 3.80666249,  7.3       ,  1.72026347,  2.39789527],
       [ 3.78418963,  4.4       ,  1.79641528,  1.94591015],
       [ 3.78418963,  6.        ,  1.71972627,  2.19722458],
       [ 3.80666249,  5.3       ,  1.5723589 ,  2.19722458],
       [ 3.73766962, 10.7       ,  2.1071783 ,  2.39789527],
       [ 3.78418963,  6.8       ,  1.80104952,  2.19722458],
       [ 3.78418963, 11.9       ,  1.97060276,  2.30258509],
       [ 3.78418963,  9.8       ,  2.0514278 ,  2.30258509],
       [ 3.78418963, 10.1       ,  2.08168901,  2.30258509],
       [ 3.80666249,  8.1       ,  1.76883196,  2.30258509],
       [ 3.80666249,  7.        ,  1.8757942 ,  2.19722458],
       [ 3.68887945, 11.8       ,  2.0483378 ,  2.39789527],
       [ 3.80666249,  6.5       ,  1.49671704,  2.07944154],
       [ 3.80666249,  7.1       ,  1.94276234,  2.19722458],
       [ 3.78418963, 11.4       ,  2.16113673,  2.30258509],
       [ 3.80666249,  7.4       ,  1.83705146,  2.07944154],
       [ 3.63758616, 10.1       ,  2.0872857 ,  2.30258509],
       [ 3.78418963, 11.1       ,  2.10936436,  2.30258509],
       [ 3.80666249,  9.        ,  1.90389697,  2.19722458],
       [ 3.78418963, 11.        ,  2.20475173,  2.30258509],
       [ 3.80666249,  3.7       ,  0.35276732,  1.94591015],
       [ 3.78418963,  6.6       ,  1.73536547,  2.19722458],
       [ 3.78418963, 11.4       ,  2.20088454,  2.30258509],
       [ 3.80666249,  8.8       ,  1.67709656,  2.07944154],
       [ 3.78418963,  6.8       ,  1.6624098 ,  2.19722458],
       [ 3.80666249,  7.4       ,  1.85910654,  2.39789527],
       [ 3.80666249,  7.1       ,  1.81661467,  2.19722458],
       [ 3.78418963, 11.2       ,  2.16859645,  2.30258509],
       [ 3.78418963,  6.        ,  1.83033895,  1.94591015],
       [ 3.78418963,  4.4       ,  1.77121657,  1.94591015],
       [ 3.78418963,  5.8       ,  1.71181398,  2.07944154],
       [ 3.80666249,  6.8       ,  1.66525098,  2.19722458],
       [ 3.80666249,  6.5       ,  1.58186114,  2.19722458],
       [ 3.78418963,  4.4       ,  1.77240674,  1.94591015],
       [ 3.80666249,  8.3       ,  1.74885197,  2.30258509],
       [ 3.78418963,  9.7       ,  1.92803709,  2.30258509],
       [ 3.78418963,  5.5       ,  1.83705146,  1.94591015],
       [ 3.80666249,  6.9       ,  1.1668938 ,  2.07944154],
       [ 3.80666249,  8.9       ,  1.60683453,  2.19722458],
       [ 3.66356165,  9.        ,  2.09568392,  2.39789527],
       [ 3.80666249,  2.        , -0.51919387,  1.79175947],
       [ 3.80666249,  7.1       ,  1.84229414,  2.19722458],
       [ 3.80666249,  8.8       ,  1.79325835,  2.39789527],
       [ 3.80666249,  8.        ,  1.72240939,  2.07944154],
       [ 3.78418963,  9.1       ,  2.1102132 ,  2.39789527],
       [ 3.78418963,  6.7       ,  1.74046617,  2.19722458],
       [ 3.78418963,  7.8       ,  1.82131827,  2.07944154],
       [ 3.78418963,  7.6       ,  1.66335775,  2.19722458],
       [ 3.78418963,  5.1       ,  1.83226141,  1.94591015],
```

```
[ 3.80666249,  3.3       , -1.2447948 ,  1.38629436],
[ 3.80666249,  5.4       ,  1.54968791,  2.19722458],
[ 3.78418963,  7.5       ,  1.73871025,  2.19722458],
[ 3.80666249,  7.7       ,  1.78355927,  2.30258509],
[ 3.80666249,  6.9       ,  1.6690267 ,  2.19722458],
[ 3.80666249,  8.        ,  1.90165966,  2.39789527],
[ 3.80666249,  8.8       ,  1.95755635,  2.19722458],
[ 3.78418963, 11.6       ,  2.16056059,  2.30258509],
[ 3.80666249,  8.8       ,  1.90031488,  2.39789527],
[ 3.80666249,  8.        ,  1.48885095,  2.07944154],
[ 3.78418963,  9.7       ,  1.83322126,  2.19722458],
[ 3.78418963,  9.4       ,  1.95529455,  2.30258509],
[ 3.80666249,  6.8       ,  1.84324441,  2.07944154],
[ 3.78418963,  6.1       ,  1.83641411,  1.94591015],
[ 3.80666249,  8.9       ,  1.8916048 ,  2.30258509],
[ 3.78418963,  9.9       ,  1.77478285,  2.07944154],
[ 3.80666249,  1.3       , -0.94160854,  1.60943791],
[ 3.80666249,  6.8       ,  1.67747032,  2.19722458],
[ 3.78418963,  9.2       ,  1.95231815,  2.30258509],
[ 3.80666249,  4.        , -1.14885351,  1.38629436],
[ 3.78418963,  6.8       ,  1.90939435,  1.94591015],
[ 3.80666249,  8.3       ,  1.66184059,  2.07944154],
[ 3.80666249,  7.8       ,  1.37826219,  2.07944154],
[ 3.80666249,  8.7       ,  1.62195919,  2.19722458],
[ 3.80666249,  8.        ,  1.74134298,  2.30258509],
[ 3.78418963,  6.1       ,  1.82985776,  1.94591015],
[ 3.63758616, 13.1       ,  2.06660956,  2.48490665],
[ 3.80666249,  5.5       ,  0.77656879,  1.94591015],
[ 3.78418963,  7.2       ,  1.89085037,  1.94591015],
[ 3.78418963, 10.7       ,  2.11106133,  2.30258509],
[ 3.78418963,  6.6       ,  1.84435192,  1.94591015],
[ 3.80666249,  7.4       ,  1.65326339,  2.30258509],
[ 3.80666249,  8.9       ,  1.9729693 ,  2.19722458],
[ 3.80666249,  4.        , -1.39030238,  1.38629436],
[ 3.78418963,  9.4       ,  2.09801793,  2.30258509],
[ 3.80666249,  6.1       ,  1.70037491,  2.19722458],
[ 3.78418963,  4.7       ,  1.68658412,  2.07944154],
[ 3.80666249,  5.7       ,  1.52453341,  2.07944154],
[ 3.80666249,  2.2       , -1.24132859,  1.38629436],
[ 3.66356165, 12.1       ,  2.07103127,  2.39789527],
[ 3.80666249,  8.2       ,  1.77172682,  2.19722458],
[ 3.80666249,  9.3       ,  1.9145669 ,  2.30258509],
[ 3.80666249,  0.9       , -0.81418551,  1.79175947],
[ 3.78418963,  6.6       ,  1.72917459,  2.19722458],
[ 3.80666249,  8.4       ,  1.97948308,  2.19722458],
[ 3.78418963,  7.5       ,  1.87502774,  1.94591015],
[ 3.78418963, 11.5       ,  2.10608347,  2.30258509],
[ 3.80666249,  8.3       ,  1.6731636 ,  2.07944154],
[ 3.78418963,  3.2       ,  0.9520439 ,  1.09861229],
[ 3.80666249,  7.3       ,  1.81319475,  2.30258509],
[ 3.78418963,  9.        ,  1.81221548,  2.07944154],
[ 3.63758616,  8.9       ,  2.09075233,  2.30258509],
[ 3.80666249,  9.2       ,  1.6654401 ,  2.19722458],
[ 3.66356165, 11.5       ,  2.06913865,  2.39789527],
[ 3.78418963,  6.5       ,  1.83449962,  1.94591015],
[ 3.80666249,  7.1       ,  1.93888553,  2.30258509],
[ 3.78418963,  6.7       ,  1.95896739,  2.07944154],
[ 3.80666249,  7.        ,  1.88494618,  2.19722458],
[ 3.78418963,  9.1       ,  1.94748034,  2.30258509],
[ 3.63758616, 13.2       ,  2.09482264,  2.48490665],
[ 3.80666249,  8.1       ,  1.82503305,  2.07944154],
```

```
[ 3.78418963,  7.4       ,  1.67747032,  2.07944154],
[ 3.78418963, 11.7       ,  2.19210036,  2.30258509],
[ 3.80666249,  5.9       ,  1.62609835,  2.19722458],
[ 3.63758616, 13.3       ,  2.09062873,  2.48490665],
[ 3.80666249,  6.5       ,  1.50207539,  2.07944154],
[ 3.78418963,  5.2       ,  1.76181559,  1.94591015],
[ 3.78418963,  7.2       ,  1.74011524,  2.19722458],
[ 3.80666249,  7.4       ,  1.67297593,  2.30258509],
[ 3.78418963,  7.6       ,  1.75785792,  2.07944154],
[ 3.78418963,  7.2       ,  1.69249147,  2.07944154],
[ 3.80666249,  3.9       , -1.43969514,  1.09861229],
[ 3.80666249,  5.5       ,  1.51072194,  2.07944154],
[ 3.80666249,  5.7       ,  1.52757689,  2.07944154],
[ 3.78418963,  7.1       ,  1.92599894,  2.07944154],
[ 3.78418963,  6.5       ,  1.83529777,  1.94591015],
[ 3.80666249,  7.4       ,  1.886615  ,  2.39789527],
[ 3.80666249,  8.3       ,  1.97962121,  2.19722458],
[ 3.80666249,  8.8       ,  1.97241297,  2.19722458],
[ 3.80666249,  6.1       ,  1.54158725,  2.07944154],
[ 3.78418963,  3.6       ,  0.90381321,  1.09861229],
[ 3.80666249,  6.8       ,  1.87793717,  2.19722458],
[ 3.80666249,  7.        ,  1.32282194,  2.07944154],
[ 3.78418963, 11.        ,  2.17962619,  2.30258509],
[ 3.80666249,  8.1       ,  1.82744831,  2.07944154],
[ 3.80666249,  7.6       ,  1.8597296 ,  2.07944154],
[ 3.80666249,  8.        ,  1.71307695,  2.30258509],
[ 3.78418963,  9.5       ,  1.79192612,  2.07944154],
[ 3.80666249,  4.8       ,  1.67241271,  2.19722458],
[ 3.63758616, 12.9       ,  2.06293099,  2.48490665],
[ 3.80666249,  8.4       ,  1.71649698,  2.19722458],
[ 3.78418963,  6.8       ,  1.83370084,  1.94591015],
[ 3.78418963,  3.9       ,  0.87003735,  1.09861229],
[ 3.73766962, 11.        ,  2.01583592,  2.39789527],
[ 3.80666249,  2.4       , -1.98777435,  1.09861229],
[ 3.80666249,  6.5       ,  1.89626962,  2.39789527],
[ 3.78418963,  7.2       ,  1.93701355,  2.07944154],
[ 3.80666249,  6.5       ,  1.56129754,  2.19722458],
[ 3.78418963,  5.5       ,  1.66921512,  2.07944154],
[ 3.80666249,  8.2       ,  1.68138641,  2.07944154],
[ 3.80666249,  6.7       ,  1.88449057,  2.19722458],
[ 3.78418963,  8.3       ,  2.11323892,  2.19722458],
[ 3.78418963,  6.5       ,  1.83274145,  1.94591015],
[ 3.80666249,  7.8       ,  1.86856618,  2.07944154],
[ 3.78418963,  8.4       ,  2.0483378 ,  2.19722458],
[ 3.80666249,  5.8       ,  1.49335343,  2.07944154],
[ 3.80666249,  6.5       ,  1.60382217,  2.19722458],
[ 3.78418963,  5.        ,  1.76404563,  1.94591015],
[ 3.80666249,  1.9       , -1.09961279,  1.60943791],
[ 3.80666249,  8.4       ,  1.69982691,  2.07944154],
[ 3.80666249,  5.1       ,  1.70201709,  2.19722458],
[ 3.80666249,  7.1       ,  1.69818112,  2.30258509],
[ 3.80666249,  7.4       ,  1.81580146,  2.19722458],
[ 3.78418963,  5.5       ,  1.67297593,  2.07944154],
[ 3.78418963, 11.5       ,  2.13794627,  2.30258509],
[ 3.80666249,  7.5       ,  1.88843236,  2.39789527],
[ 3.80666249,  4.2       , -1.37042101,  1.09861229],
[ 3.80666249,  8.6       ,  1.68917288,  2.07944154],
[ 3.78418963,  7.3       ,  1.85957387,  1.94591015],
[ 3.78418963,  9.1       ,  1.9139771 ,  2.19722458],
[ 3.80666249,  7.7       ,  1.92861865,  2.30258509],
[ 3.80666249,  5.8       ,  1.6910179 ,  2.19722458],
```

```
       [ 3.80666249,  8.3       ,  1.75768549,  2.19722458],
       [ 3.80666249,  8.1       ,  1.68268837,  2.30258509],
       [ 3.80666249,  2.4       , -0.13124829,  1.94591015],
       [ 3.78418963, 10.8       ,  2.15547623,  2.30258509],
       [ 3.80666249,  7.4       ,  1.80055402,  2.39789527],
       [ 3.80666249,  7.8       ,  1.79042525,  2.19722458],
       [ 3.80666249,  9.4       ,  1.93398215,  2.19722458],
       [ 3.78418963,  6.6       ,  1.7344834 ,  2.19722458],
       [ 3.78418963,  7.9       ,  1.99470031,  2.19722458],
       [ 3.80666249,  8.9       ,  1.60100243,  2.19722458],
       [ 3.80666249,  6.9       ,  1.68861871,  2.30258509],
       [ 3.80666249,  7.6       ,  1.57856625,  2.19722458],
       [ 3.78418963,  8.8       ,  1.85301135,  2.19722458],
       [ 3.78418963,  6.7       ,  1.70638312,  2.07944154],
       [ 3.63758616, 14.4       ,  2.06420099,  2.48490665],
       [ 3.78418963,  7.4       ,  1.946053  ,  2.07944154],
       [ 3.78418963,  6.1       ,  1.73888598,  2.19722458],
       [ 3.78418963,  7.1       ,  1.82196531,  2.07944154],
       [ 3.78418963,  9.3       ,  2.16136709,  2.19722458],
       [ 3.66356165, 13.3       ,  2.06863334,  2.39789527],
       [ 3.80666249,  6.6       ,  1.63139508,  2.19722458],
       [ 3.80666249,  7.4       ,  1.64634827,  2.19722458],
       [ 3.80666249,  7.4       ,  1.65326339,  2.07944154],
       [ 3.78418963,  8.4       ,  1.89536845,  2.19722458],
       [ 3.78418963,  6.5       ,  1.83210135,  1.94591015],
       [ 3.80666249,  7.8       ,  1.83146084,  2.07944154],
       [ 3.78418963,  6.6       ,  1.82985776,  1.94591015],
       [ 3.78418963,  7.5       ,  1.7406416 ,  2.07944154],
       [ 3.80666249,  6.1       ,  1.64557696,  2.19722458],
       [ 3.71357207, 11.4       ,  2.03182569,  2.39789527],
       [ 3.78418963,  8.8       ,  2.18019144,  2.19722458],
       [ 3.78418963,  6.8       ,  1.71757457,  2.07944154],
       [ 3.78418963,  6.4       ,  1.83194126,  1.94591015],
       [ 3.80666249,  8.4       ,  1.8454582 ,  2.39789527],
       [ 3.80666249,  7.2       ,  1.43413169,  2.07944154],
       [ 3.78418963,  6.8       ,  1.72490716,  2.19722458],
       [ 3.78418963,  4.2       ,  1.77155676,  1.94591015],
       [ 3.78418963,  5.8       ,  1.69799809,  2.19722458],
       [ 3.78418963,  4.9       ,  1.76951386,  1.94591015],
       [ 3.80666249,  8.5       ,  1.85442123,  2.39789527],
       [ 3.78418963, 10.7       ,  1.9645917 ,  2.39789527],
       [ 3.80666249,  5.7       ,  1.50318811,  2.07944154],
       [ 3.80666249,  8.3       ,  1.87333946,  2.39789527],
       [ 3.80666249,  5.7       ,  1.51072194,  2.07944154],
       [ 3.78418963, 10.1       ,  2.19265861,  2.30258509],
       [ 3.78418963,  4.2       ,  1.81123525,  1.94591015],
       [ 3.63758616, 11.1       ,  2.09666732,  2.39789527],
       [ 3.80666249,  1.9       , -1.42295835,  1.38629436],
       [ 3.73766962, 10.4       ,  1.99823149,  2.30258509],
       [ 3.78418963,  7.6       ,  1.91471429,  2.07944154],
       [ 3.78418963,  9.4       ,  1.95868534,  2.39789527],
       [ 3.80666249,  8.2       ,  1.71972627,  2.07944154],
       [ 3.80666249,  9.        ,  1.96164288,  2.19722458],
       [ 3.78418963,  5.        ,  1.67765715,  2.07944154]])
```

In [247]:

```
# Fitting Optimized Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
omr = LinearRegression().fit(X_train[:, 0:4], y_train)
```

In [248]:

```
# Getting parameters
omr.coef_, omr.intercept_
```

Out[248]:

```
(array([-83.62268193,   0.52170896,  21.84091709,  -4.86996451]),
 291.436212016663)
```

In [249]:

```
# Predicting the Test set results
y_pred = omr.predict(X_test[:, 0:4])
```

In [250]:

```
# Coefficient of determination R^2
omr.score(X_train[:, 0:4], y_train), omr.score(X_test[:, 0:4], y_test)
```

Out[250]:

```
(0.3479422499743793, 0.4369496099590431)
```

In [251]:

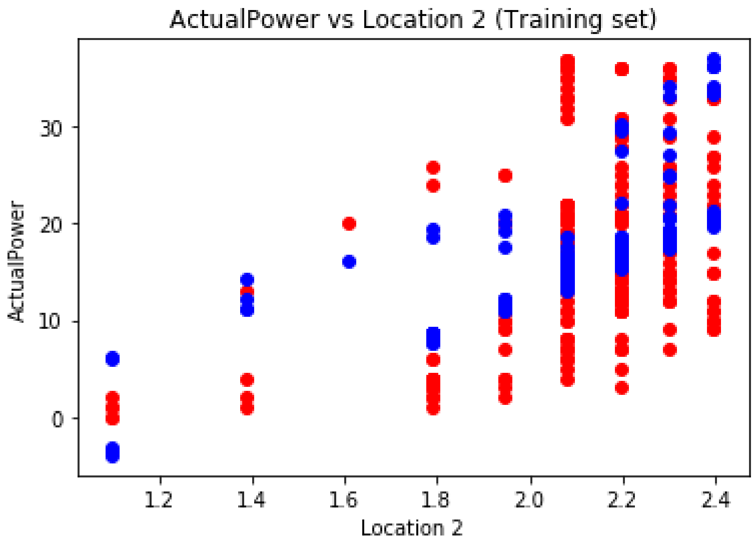
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, omr.predict(X_train[:, 0:4])), mean_squared_error(y_test, y
_pred)
```

Out[251]:

```
(62.49427741881587, 54.64095902341732)
```

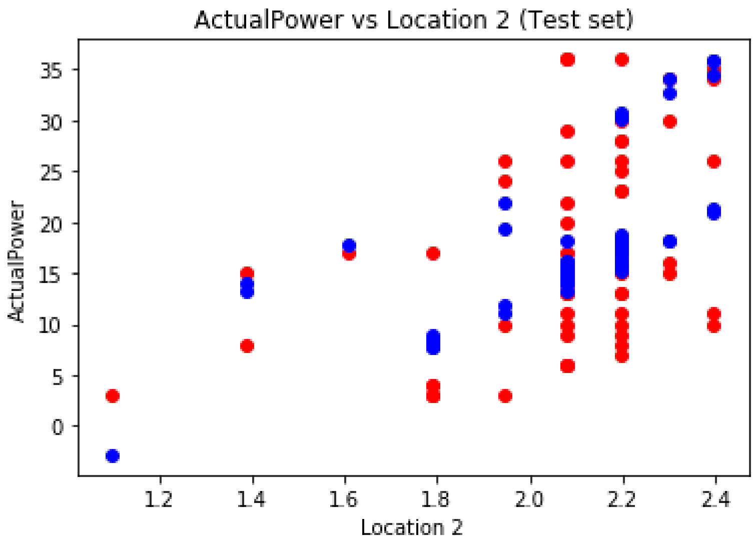
In [252]:

```
# Visualising the Training set results
plt.scatter(X_train[:,2], y_train, color = 'red')
plt.plot(X_train[:,2], omr.predict(X_train[:, 0:4]), 'bo')
plt.title('ActualPower vs Location 2 (Training set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [253]:

```
# Visualising the Test set results
plt.scatter(X_test[:,2], y_test, color = 'red')
plt.plot(X_test[:,2], omr.predict(X_test[:, 0:4]), 'bo')
plt.title('ActualPower vs Location 2 (Test set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



Regression Tree & Random Forest

In [254]:

```
df_log
```

Out[254]:

	ActualPower	Max Capacity	Location 1	Location 2	Location 3	Location 4
0	0.0	3.806662	3.9	1.098612	-1.439695	1.098612
1	0.0	3.806662	4.2	1.098612	-1.370421	1.098612
2	1.0	3.806662	4.0	1.386294	-1.390302	1.386294
3	2.0	3.806662	4.0	1.386294	-1.148854	1.386294
4	4.0	3.806662	3.3	1.386294	-1.244795	1.386294
...	...	...	...	...	...	...
289	1.0	3.784190	4.0	1.791759	1.746239	1.791759
290	1.0	3.784190	3.2	1.098612	0.952044	1.098612
291	2.0	3.784190	3.6	1.098612	0.903813	1.098612
292	1.0	3.784190	3.9	1.098612	0.870037	1.098612
293	3.0	3.784190	4.3	1.098612	0.837680	1.098612

294 rows × 6 columns

In [255]:

```
# Splitting the dataset into the Training set and Test set
X = df_log.iloc[:, :-1].values
y = df_log.iloc[:, 5].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0
)
```

In [256]:

```
# Fitting Tree to the Training set (Location 2)
from sklearn.tree import DecisionTreeRegressor
sdt = DecisionTreeRegressor(max_leaf_nodes = 10).fit(X_train[:, 2:3], y_train)
```

In [257]:

```
# Predicting the Test set results
y_pred = sdt.predict(X_test[:, 2:3])
```

In [258]:

```
# Coefficient of determination R^2 (Коэффициент детерминации значительно выше, чем в пр
едыдущих моделях. Regression Tree уже есть смысл использовать)
sdt.score(X_train[:, 2:3], y_train), sdt.score(X_test[:, 2:3], y_test)
```

Out[258]:

```
(0.761048882782419, 0.7198186570370979)
```

In [259]:

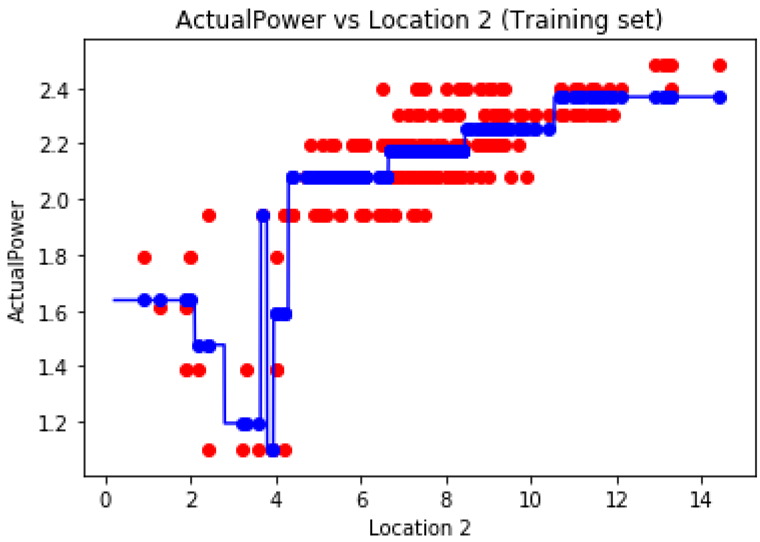
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, sdt.predict(X_train[:, 2:3])), mean_squared_error(y_test, y
_pred)
```

Out[259]:

```
(0.015329857642374097, 0.028014451710926282)
```

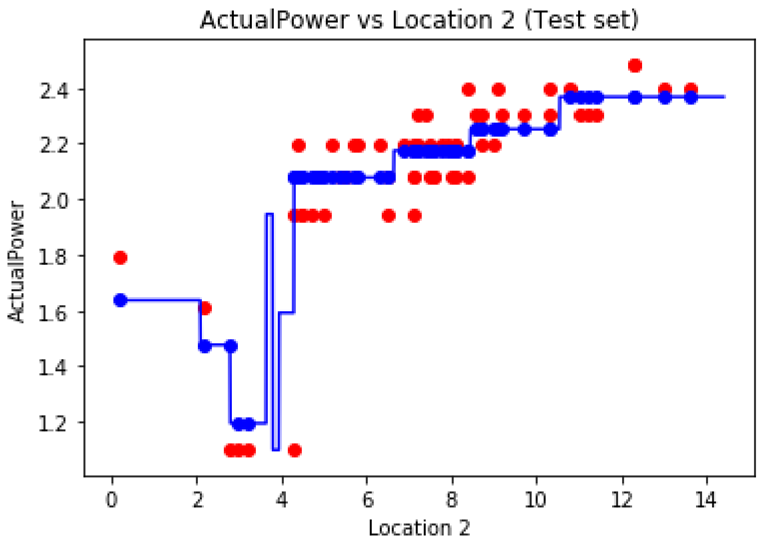
In [260]:

```
# Visualising the Training set results
X_grid = np.arange(min(X[:, 2:3]), max(X[:, 2:3]), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.plot(X_grid, sdt.predict(X_grid), color = 'blue')
plt.scatter(X_train[:,2], y_train, color = 'red')
plt.plot(X_train[:,2], sdt.predict(X_train[:, 2:3]), 'bo')
plt.title('ActualPower vs Location 2 (Training set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [261]:

```
# Visualising the Test set results
X_grid = np.arange(min(X[:, 2:3]), max(X[:, 2:3]), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.plot(X_grid, sdt.predict(X_grid), color = 'blue')
plt.scatter(X_test[:,2], y_test, color = 'red')
plt.plot(X_test[:,2], sdt.predict(X_test[:, 2:3]), 'bo')
plt.title('ActualPower vs Location 2 (Test set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [262]:

```
# Fitting Tree to the Training set
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor().fit(X_train, y_train)
```

In [263]:

```
# Predicting the Test set results
y_pred = dt.predict(X_test)
```

In [264]:

```
# Coefficient of determination R^2
dt.score(X_train, y_train), dt.score(X_test, y_test)
```

Out[264]:

```
(1.0, 0.9480800376972474)
```

In [265]:

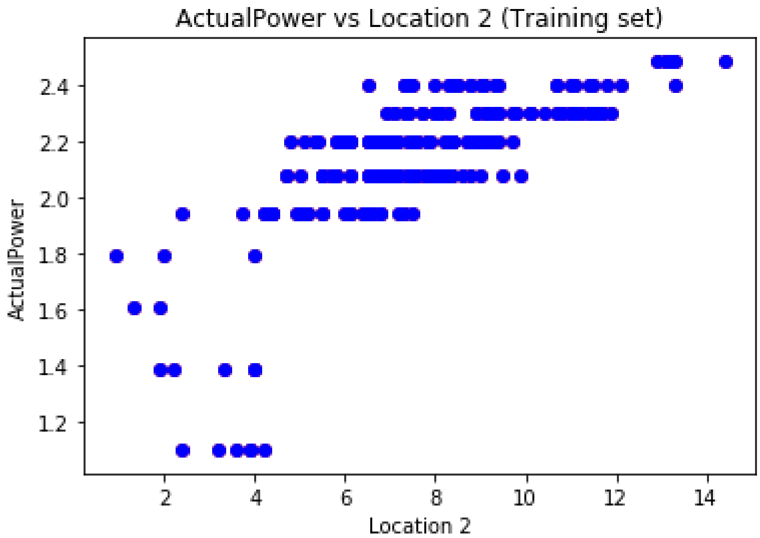
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, dt.predict(X_train)), mean_squared_error(y_test, y_pred)
```

Out[265]:

```
(7.48998253095482e-32, 0.005191313816195695)
```

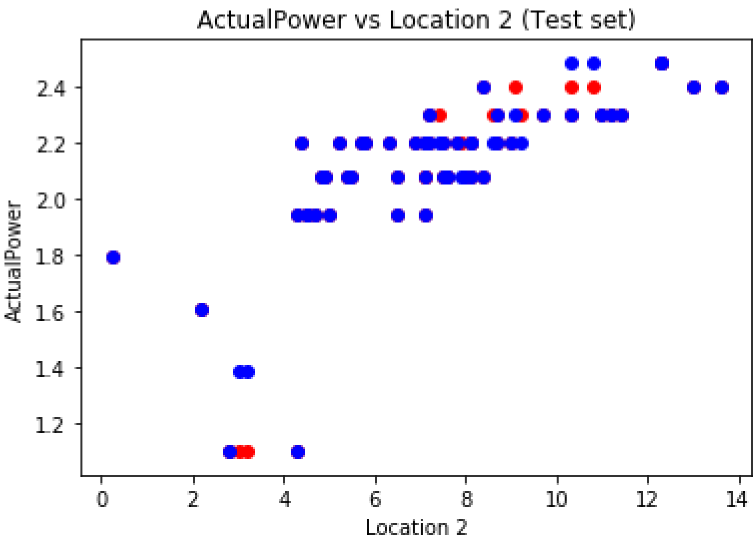
In [266]:

```
# Visualising the Training set results
plt.scatter(X_train[:,2], y_train, color = 'red')
plt.plot(X_train[:,2], dt.predict(X_train), 'bo')
plt.title('ActualPower vs Location 2 (Training set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [267]:

```
# Visualising the Test set results
plt.scatter(X_test[:,2], y_test, color = 'red')
plt.plot(X_test[:,2], dt.predict(X_test), 'bo')
plt.title('ActualPower vs Location 2 (Test set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [268]:

```
# Fitting Random Forest to the Training set
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators = 10, random_state = 0).fit(X_train, y_train)
```

In [269]:

```
# Predicting the Test set results
y_pred = rf.predict(X_test)
```

In [270]:

```
# Coefficient of determination R^2
rf.score(X_train, y_train), rf.score(X_test, y_test)
```

Out[270]:

```
(0.9872288937522067, 0.9312037957913394)
```

In [271]:

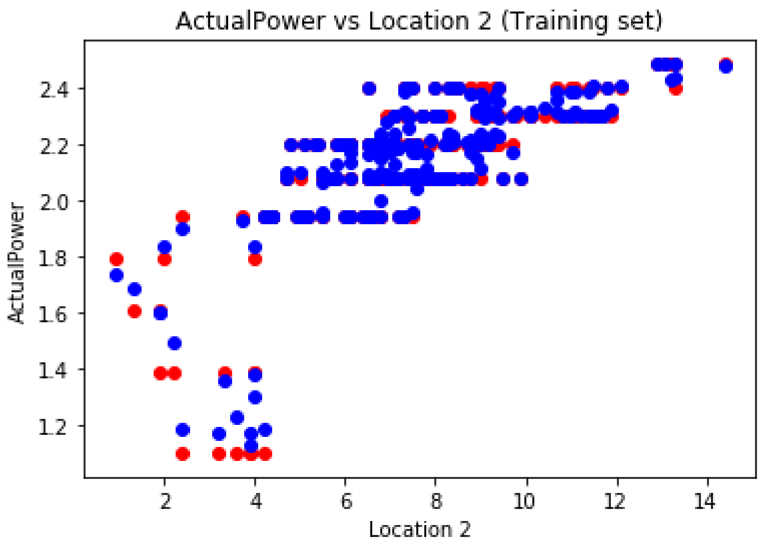
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, rf.predict(X_train)), mean_squared_error(y_test, y_pred)
```

Out[271]:

```
(0.0008193275804441426, 0.006878716192582933)
```

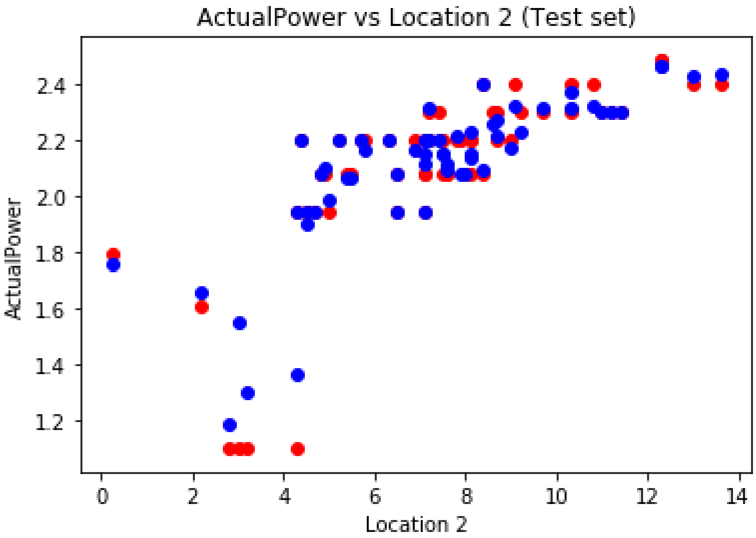
In [272]:

```
# Visualising the Training set results
plt.scatter(X_train[:,2], y_train, color = 'red')
plt.plot(X_train[:,2], rf.predict(X_train), 'bo')
plt.title('ActualPower vs Location 2 (Training set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [273]:

```
# Visualising the Test set results
plt.scatter(X_test[:,2], y_test, color = 'red')
plt.plot(X_test[:,2], rf.predict(X_test), 'bo')
plt.title('ActualPower vs Location 2 (Test set)')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



Regression Neural Network

In [274]:

```
df_log
```

Out[274]:

	ActualPower	Max Capacity	Location 1	Location 2	Location 3	Location 4
0	0.0	3.806662	3.9	1.098612	-1.439695	1.098612
1	0.0	3.806662	4.2	1.098612	-1.370421	1.098612
2	1.0	3.806662	4.0	1.386294	-1.390302	1.386294
3	2.0	3.806662	4.0	1.386294	-1.148854	1.386294
4	4.0	3.806662	3.3	1.386294	-1.244795	1.386294
...	...	...	...	...	...	...
289	1.0	3.784190	4.0	1.791759	1.746239	1.791759
290	1.0	3.784190	3.2	1.098612	0.952044	1.098612
291	2.0	3.784190	3.6	1.098612	0.903813	1.098612
292	1.0	3.784190	3.9	1.098612	0.870037	1.098612
293	3.0	3.784190	4.3	1.098612	0.837680	1.098612

294 rows × 6 columns

In [275]:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
dfsc = sc.fit_transform(df)
df_log['ActualPower'] = dfsc[:,0]
df_log['Max Capacity'] = dfsc[:,1]
df_log['Location 1'] = dfsc[:,2]
df_log['Location 2'] = dfsc[:,3]
df_log['Location 3'] = dfsc[:,4]
df_log['Location 4'] = dfsc[:,5]
```

In [276]:

```
# Cheking correlations
df_log.corr()
```

Out[276]:

	ActualPower	Max Capacity	Location 1	Location 2	Location 3	Location 4
ActualPower	1.000000	-0.407037	0.379332	0.425873	0.176696	0.432648
Max Capacity	-0.407037	1.000000	-0.537528	-0.232178	-0.426030	-0.381246
Location 1	0.379332	-0.537528	1.000000	0.652382	0.796576	0.777234
Location 2	0.425873	-0.232178	0.652382	1.000000	0.587584	0.922757
Location 3	0.176696	-0.426030	0.796576	0.587584	1.000000	0.762429
Location 4	0.432648	-0.381246	0.777234	0.922757	0.762429	1.000000

In [277]:

```
# Splitting the dataset into the Training set and Test set
X = df_log.iloc[:, 1:6].values
y = df_log.iloc[:, 0].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0
)
```

In [278]:

```
# Install Tensorflow
# Install Keras
# Importing the Keras libraries and packages
# !pip3 install keras
# !pip install tensorflow
import keras
from keras.models import Sequential
from keras.layers import Dense
```

In [279]:

```
# Initialising the ANN
rnn = Sequential()

# Adding the input layer and the first hidden layer
rnn.add(Dense(units = 6, activation = 'tanh', input_dim = 5))

# Adding the second hidden layer
rnn.add(Dense(units = 6, activation = 'tanh'))

# Adding the output layer
rnn.add(Dense(units = 1, activation = 'linear'))

# Compiling the ANN
rnn.compile(optimizer='adam', loss='mean_squared_error', metrics = ['accuracy'])
```

In [280]:

```
# Fitting the ANN to the Training set
rnn.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

```
Epoch 1/100
24/24 [==============================] - 0s 1ms/step - loss: 1.1181 - accu
racy: 0.0000e+00
Epoch 2/100
24/24 [==============================] - 0s 956us/step - loss: 1.0320 - ac
curacy: 0.0000e+00
Epoch 3/100
24/24 [==============================] - 0s 1ms/step - loss: 0.9558 - accu
racy: 0.0000e+00
Epoch 4/100
24/24 [==============================] - 0s 1ms/step - loss: 0.9079 - accu
racy: 0.0000e+00
Epoch 5/100
24/24 [==============================] - 0s 1ms/step - loss: 0.8636 - accu
racy: 0.0000e+00
Epoch 6/100
24/24 [==============================] - 0s 997us/step - loss: 0.8325 - ac
curacy: 0.0000e+00
Epoch 7/100
24/24 [==============================] - 0s 1ms/step - loss: 0.8071 - accu
racy: 0.0000e+00
Epoch 8/100
24/24 [==============================] - 0s 1ms/step - loss: 0.7853 - accu
racy: 0.0000e+00
Epoch 9/100
24/24 [==============================] - 0s 997us/step - loss: 0.7674 - ac
curacy: 0.0000e+00
Epoch 10/100
24/24 [==============================] - 0s 2ms/step - loss: 0.7496 - accu
racy: 0.0000e+00
Epoch 11/100
24/24 [==============================] - 0s 1ms/step - loss: 0.7349 - accu
racy: 0.0000e+00
Epoch 12/100
24/24 [==============================] - 0s 1ms/step - loss: 0.7197 - accu
racy: 0.0000e+00
Epoch 13/100
24/24 [==============================] - 0s 1ms/step - loss: 0.7048 - accu
racy: 0.0000e+00
Epoch 14/100
24/24 [==============================] - 0s 1ms/step - loss: 0.6922 - accu
racy: 0.0000e+00
Epoch 15/100
24/24 [==============================] - 0s 1ms/step - loss: 0.6799 - accu
racy: 0.0000e+00
Epoch 16/100
24/24 [==============================] - 0s 1ms/step - loss: 0.6692 - accu
racy: 0.0000e+00
Epoch 17/100
24/24 [==============================] - 0s 997us/step - loss: 0.6555 - ac
curacy: 0.0000e+00
Epoch 18/100
24/24 [==============================] - 0s 1ms/step - loss: 0.6445 - accu
racy: 0.0000e+00
Epoch 19/100
24/24 [==============================] - 0s 997us/step - loss: 0.6354 - ac
curacy: 0.0000e+00
Epoch 20/100
24/24 [==============================] - 0s 956us/step - loss: 0.6257 - ac
curacy: 0.0000e+00
Epoch 21/100
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.6176 - accu
racy: 0.0000e+00
Epoch 22/100
24/24 [==============================] - 0s 1ms/step - loss: 0.6094 - accu
racy: 0.0000e+00
Epoch 23/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5988 - accu
racy: 0.0000e+00
Epoch 24/100
24/24 [==============================] - 0s 956us/step - loss: 0.5904 - ac
curacy: 0.0000e+00
Epoch 25/100
24/24 [==============================] - 0s 997us/step - loss: 0.5840 - ac
curacy: 0.0000e+00
Epoch 26/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5751 - accu
racy: 0.0000e+00
Epoch 27/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5685 - accu
racy: 0.0000e+00
Epoch 28/100
24/24 [==============================] - 0s 2ms/step - loss: 0.5606 - accu
racy: 0.0000e+00
Epoch 29/100
24/24 [==============================] - 0s 956us/step - loss: 0.5564 - ac
curacy: 0.0000e+00
Epoch 30/100
24/24 [==============================] - 0s 997us/step - loss: 0.5520 - ac
curacy: 0.0000e+00
Epoch 31/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5429 - accu
racy: 0.0000e+00
Epoch 32/100
24/24 [==============================] - 0s 997us/step - loss: 0.5367 - ac
curacy: 0.0000e+00
Epoch 33/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5305 - accu
racy: 0.0000e+00
Epoch 34/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5288 - accu
racy: 0.0000e+00
Epoch 35/100
24/24 [==============================] - 0s 997us/step - loss: 0.5212 - ac
curacy: 0.0000e+00
Epoch 36/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5172 - accu
racy: 0.0000e+00
Epoch 37/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5107 - accu
racy: 0.0000e+00
Epoch 38/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5047 - accu
racy: 0.0000e+00
Epoch 39/100
24/24 [==============================] - 0s 1ms/step - loss: 0.5023 - accu
racy: 0.0000e+00
Epoch 40/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4960 - accu
racy: 0.0000e+00
Epoch 41/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4925 - accu
```

```
racy: 0.0000e+00
Epoch 42/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4862 - accu
racy: 0.0000e+00
Epoch 43/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4822 - accu
racy: 0.0000e+00
Epoch 44/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4770 - accu
racy: 0.0000e+00
Epoch 45/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4732 - accu
racy: 0.0000e+00
Epoch 46/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4697 - accu
racy: 0.0000e+00
Epoch 47/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4668 - accu
racy: 0.0000e+00
Epoch 48/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4627 - accu
racy: 0.0000e+00
Epoch 49/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4598 - accu
racy: 0.0000e+00
Epoch 50/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4554 - accu
racy: 0.0000e+00
Epoch 51/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4533 - accu
racy: 0.0000e+00
Epoch 52/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4493 - accu
racy: 0.0000e+00
Epoch 53/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4450 - accu
racy: 0.0000e+00
Epoch 54/100
24/24 [==============================] - 0s 997us/step - loss: 0.4438 - ac
curacy: 0.0000e+00
Epoch 55/100
24/24 [==============================] - 0s 997us/step - loss: 0.4385 - ac
curacy: 0.0000e+00
Epoch 56/100
24/24 [==============================] - 0s 997us/step - loss: 0.4363 - ac
curacy: 0.0000e+00
Epoch 57/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4339 - accu
racy: 0.0000e+00
Epoch 58/100
24/24 [==============================] - 0s 997us/step - loss: 0.4335 - ac
curacy: 0.0000e+00
Epoch 59/100
24/24 [==============================] - 0s 956us/step - loss: 0.4307 - ac
curacy: 0.0000e+00
Epoch 60/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4259 - accu
racy: 0.0000e+00
Epoch 61/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4260 - accu
racy: 0.0000e+00
```

```
Epoch 62/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4217 - accu
racy: 0.0000e+00
Epoch 63/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4212 - accu
racy: 0.0000e+00
Epoch 64/100
24/24 [==============================] - 0s 956us/step - loss: 0.4189 - ac
curacy: 0.0000e+00
Epoch 65/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4206 - accu
racy: 0.0000e+00
Epoch 66/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4160 - accu
racy: 0.0000e+00
Epoch 67/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4122 - accu
racy: 0.0000e+00
Epoch 68/100
24/24 [==============================] - 0s 997us/step - loss: 0.4122 - ac
curacy: 0.0000e+00
Epoch 69/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4092 - accu
racy: 0.0000e+00
Epoch 70/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4070 - accu
racy: 0.0000e+00
Epoch 71/100
24/24 [==============================] - 0s 997us/step - loss: 0.4058 - ac
curacy: 0.0000e+00
Epoch 72/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4043 - accu
racy: 0.0000e+00
Epoch 73/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4046 - accu
racy: 0.0000e+00
Epoch 74/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4021 - accu
racy: 0.0000e+00
Epoch 75/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4004 - accu
racy: 0.0000e+00
Epoch 76/100
24/24 [==============================] - 0s 1ms/step - loss: 0.4003 - accu
racy: 0.0000e+00
Epoch 77/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3997 - accu
racy: 0.0000e+00
Epoch 78/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3984 - accu
racy: 0.0000e+00
Epoch 79/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3964 - accu
racy: 0.0000e+00
Epoch 80/100
24/24 [==============================] - 0s 997us/step - loss: 0.3953 - ac
curacy: 0.0000e+00
Epoch 81/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3937 - accu
racy: 0.0000e+00
Epoch 82/100
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.3927 - accu
racy: 0.0000e+00
Epoch 83/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3924 - accu
racy: 0.0000e+00
Epoch 84/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3920 - accu
racy: 0.0000e+00
Epoch 85/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3903 - accu
racy: 0.0000e+00
Epoch 86/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3886 - accu
racy: 0.0000e+00
Epoch 87/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3891 - accu
racy: 0.0000e+00
Epoch 88/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3876 - accu
racy: 0.0000e+00
Epoch 89/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3879 - accu
racy: 0.0000e+00
Epoch 90/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3866 - accu
racy: 0.0000e+00
Epoch 91/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3867 - accu
racy: 0.0000e+00
Epoch 92/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3858 - accu
racy: 0.0000e+00
Epoch 93/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3837 - accu
racy: 0.0000e+00
Epoch 94/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3832 - accu
racy: 0.0000e+00
Epoch 95/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3826 - accu
racy: 0.0000e+00
Epoch 96/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3827 - accu
racy: 0.0000e+00
Epoch 97/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3823 - accu
racy: 0.0000e+00
Epoch 98/100
24/24 [==============================] - 0s 1ms/step - loss: 0.3800 - accu
racy: 0.0000e+00
Epoch 99/100
24/24 [==============================] - 0s 2ms/step - loss: 0.3832 - accu
racy: 0.0000e+00
Epoch 100/100
24/24 [==============================] - 0s 997us/step - loss: 0.3800 - ac
curacy: 0.0000e+00
```

Out[280]:

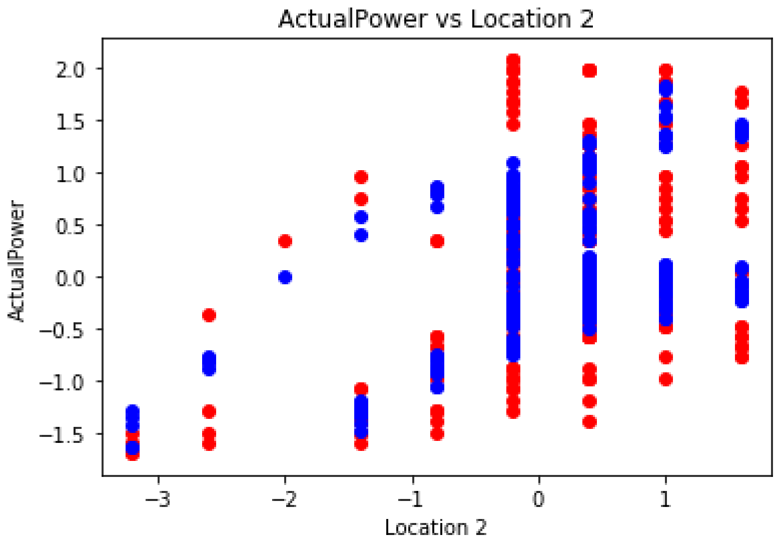
```
<tensorflow.python.keras.callbacks.History at 0x1c4a6d183a0>
```

In [281]:

```
# Predicting the Test set results
y_pred = rnn.predict(X_test)
```

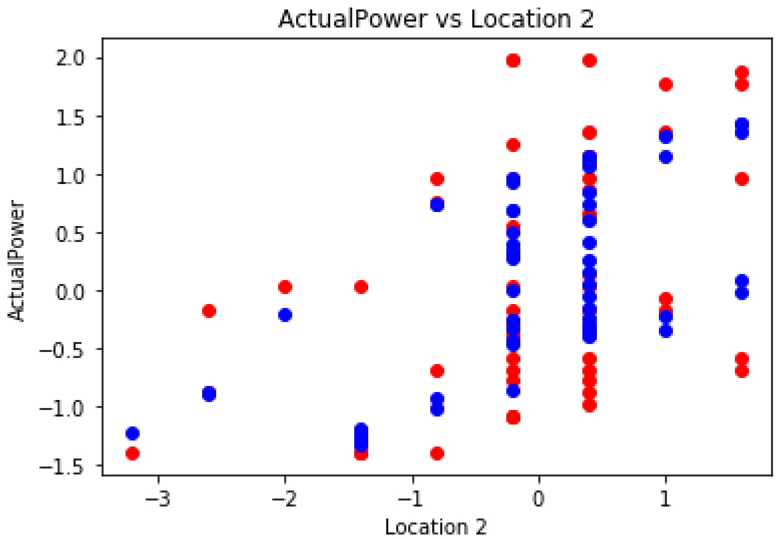
In [282]:

```
# Visualising the Training set results
plt.scatter(X_train[:,2], y_train, color = 'red')
plt.plot(X_train[:,2], rnn.predict(X_train), 'bo')
plt.title('ActualPower vs Location 2')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [283]:

```
# Visualising the Test set results
plt.scatter(X_test[:,2], y_test, color = 'red')
plt.plot(X_test[:,2], rnn.predict(X_test), 'bo')
plt.title('ActualPower vs Location 2')
plt.xlabel('Location 2')
plt.ylabel('ActualPower')
plt.show()
```



In [284]:

```
# Mean Squared Error
from sklearn.metrics import mean_squared_error
y_pred_test = rnn.predict(X_test)
print("Mean Squared Error for Test Set:")
rnn1_metrics.append(mean_squared_error(y_pred_test, y_test))
mean_squared_error(y_pred_test, y_test)
```

```
Mean Squared Error for Test Set:
```

Out[284]:

```
0.490490753697292
```

In [285]:

```
y_pred_train = rnn.predict(X_train)
print("Mean Squared Error for Train Set:")
rnn1_metrics.append(mean_squared_error(y_pred_train, y_train))
mean_squared_error(y_pred_train, y_train)
```

```
Mean Squared Error for Train Set:
```

Out[285]:

```
0.3771361099444928
```

In [286]:

```
# Как вывод наблюдаем, что по метрике MSE для тестовой виборки лучшей является модель д
ерева решений.
# Также отметим, что достаточно неплохой оказалась регрессионная модель нейронных сете
й, однако она уступает по тестовой MSE моделям Regression Tree и Random Forest
# Таким образом, лучшими моделями по критерию MSE для тестовой выборки являются Regress
ion Tree, Random Forest и Regression Neural Network.

# Ниже расположены краткие оценки моделей по убыванию.

# Regression Tree
# Coefficient of determination R^2
# (1.0, 0.9470117847884222)
# Mean squared error
# (7.48998253095482e-32, 0.0052981250664126695)

# Random Forest
# Coefficient of determination R^2
# (0.9872288937522067, 0.9312037957913394)
# Mean squared error
# (0.0008193275804441426, 0.006878716192582933)

# Regression Neural Network
# Mean squared error
# (0.45263429242919107, 0.2759985798707595)

# Multiple Linear Regression
# Coefficient of determination R^2
# (0.3510663176818859, 0.44470012975574835)
# Mean squared error
# (62.194861678445626, 53.88881349237319)

# Backward Elimination with p-values
# Coefficient of determination R^2
# (0.3479422499743793, 0.4369496099590431)
# Mean squared error
# (62.49427741881587, 54.64095902341732)

# Simple Linear Regression
# Coefficient of determination R^2
# (0.1908749365272442, 0.17233498442617246)
# Mean squared error
# (77.54786471167098, 80.32036031055034)

# Polynomial Regression
# Coefficient of determination R^2
# (0.19087526134588262, 0.1721999186006038)
# Mean squared error
# (77.54783358052332, 80.33346770976522)
```