CSC 5741: Lecture #07—Linear Regression, Classification and Clustering

Lighton Phiri
<lighton.phiri@unza.zm>

May 7 2019

Contents

Introduction **1**
Importing Libraries and Modules 1
Datasets 2
JCTR BnB 2
Boston Housing Prices 2
Simple Linear Regression 3
Example #1: JCTR Lusaka BnB 3
Exercise 10
Multiple Linear Regression 10
Example #1: JCTR Zambian Towns BnB 10
Excercise #2 18
Single Variable Usnig K Fold Cross-Validation 20

Introduction

During these “hands-on” activities, we briefly look at examplers of linear regression. Specifically, we shall look at Simple Linear Regression (Univariate Variables) and Multiple Linear Regression (One Hot Encoding).

In all instances, you are encouraged to make reference to online Python documentation and documentation for specific libraries. You are also encouraged to look up and explore other libraries, especially as you work towards the Mini Projects.

Importing Libraries and Modules

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

from datetime import datetime
from dateutil.parser import parse
from IPython.core.interactiveshell import InteractiveShell
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

plt.style.use("ggplot")

InteractiveShell.ast_node_interactivity = "all"
np.set_printoptions(suppress=True)
pd.set_option('display.latex.repr', True)
pd.set_option('display.latex.longtable', True)
```

Datasets

- We shall primarily work with the Jesuit Centre for Theological Reflection (JCTR) Basic Needs Basket (BnB) historical amounts compiled monthly, during the period November 2016 to April 2018.
- We shall also make use of "Toy Datasets" available in the Scikit Learn library.

JCTR BnB

[2]:
```
# Lusaka BnB amounts
!head -n 5 db-jctr-bnb-lusaka.csv | cat -n
```

```
     1	Town|Date|Amount
     2	Lusaka|Nov 16|5,005.14
     3	Lusaka|Dec 16|4,976.67
     4	Lusaka|Jan 17|4,935.46
     5	Lusaka|Feb 17|4,918.76
```

[3]:
```
# Zambian towns BnB amounts
!head -n 5 db-jctr-bnb-zambia.csv | cat -n
```

```
     1	Town|Date|Amount
     2	Chinsali|Nov 16|2837.4
     3	Chinsali|Dec 16|2788.35
     4	Chinsali|Jan 17|2728
     5	Chinsali|Feb 17|2740.65
```

Boston Housing Prices

[4]:
```
datasets.load_boston()["data"]
```

[4]:
```
array([[  0.00632,  18.     ,   2.31   , ...,  15.3    , 396.9    ,
          4.98   ],
       [  0.02731,   0.     ,   7.07   , ...,  17.8    , 396.9    ,
          9.14   ],
       [  0.02729,   0.     ,   7.07   , ...,  17.8    , 392.83   ,
          4.03   ],
       ...,
       [ 0.06076,  0.     , 11.93   , ..., 21.     , 396.9   ,
          5.64   ],
       [ 0.10959,  0.     , 11.93   , ..., 21.     , 393.45  ,
          6.48   ],
       [ 0.04741,  0.     , 11.93   , ..., 21.     , 396.9   ,
          7.88   ]])
```

Simple Linear Regression

Example #1: JCTR Lusaka BnB

Import Dataset Using Pandas

[5]:

```
# Convert CSV dataset into pandas DataFrame
var_jctr_bnb_lusaka = pd.read_csv("db-jctr-bnb-lusaka.csv", sep="|")
```

[6]:

```
var_jctr_bnb_lusaka.head(2)
```

[6]:

	Town	Date	Amount
0	Lusaka	Nov 16	5,005.14
1	Lusaka	Dec 16	4,976.67

Data Cleaning

[7]:

```
# Checking for NULL values
var_jctr_bnb_lusaka.isnull().values.any()
```

[7]:

```
False
```

[8]:

```
# Convert date strings to date objects
var_jctr_bnb_lusaka["Date"][0]
type(var_jctr_bnb_lusaka["Date"][0])

# df['col'] = pd.to_datetime(df['col'])
# Please see documentation [1, 2] for details on working with date strings
# [1] http://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html
# [1] https://docs.python.org/3/library/datetime.html
pd.to_datetime(var_jctr_bnb_lusaka["Date"], format="%b %y").apply(lambda var_x: var_x.
 ↪toordinal())

# Replace date string with equivalent date objects
var_jctr_bnb_lusaka["DateX"] = pd.to_datetime(var_jctr_bnb_lusaka["Date"], format="%b␣
 ↪%y").apply(lambda var_x: var_x.toordinal())
```

[8]:

```
'Nov 16'
```

[8]:

```
str
```

[8]:

	Date
0	736269
1	736299
2	736330
3	736361
4	736389
5	736420
6	736450
7	736481
8	736511
9	736542
10	736573
11	736603
12	736634
13	736664
14	736695
15	736726
16	736754
17	736785

[9]:

```
# Convert amount strings to numeric values
var_jctr_bnb_lusaka["Amount"][0]
type(var_jctr_bnb_lusaka["Amount"][0])

# Replace "," with empty string and convert to float
var_jctr_bnb_lusaka["Amount"].str.replace(",", "").astype("float")

# Replace amount strings with equivalent numeric objects
var_jctr_bnb_lusaka["Amount"] = var_jctr_bnb_lusaka["Amount"].str.replace(",", "").
 ↪astype("float")
```

[9]:

```
'5,005.14'
```

[9]:

```
str
```

[9]:

	Amount
0	5005.14
1	4976.67
2	4935.46
3	4918.76
4	5017.09
5	4973.03
6	4952.69
7	4958.52
8	4859.35
9	4928.37
10	4883.57

Continued on next page

	Amount
11	4869.47
12	4924.54
13	4957.47
14	5229.14
15	5385.42
16	5574.81
17	5433.04

[10]:

```
var_jctr_bnb_lusaka
```

[10]:

	Town	Date	Amount	DateX
0	Lusaka	Nov 16	5005.14	736269
1	Lusaka	Dec 16	4976.67	736299
2	Lusaka	Jan 17	4935.46	736330
3	Lusaka	Feb 17	4918.76	736361
4	Lusaka	Mar 17	5017.09	736389
5	Lusaka	Apr 17	4973.03	736420
6	Lusaka	May 17	4952.69	736450
7	Lusaka	Jun 17	4958.52	736481
8	Lusaka	Jul 17	4859.35	736511
9	Lusaka	Aug 17	4928.37	736542
10	Lusaka	Sep 17	4883.57	736573
11	Lusaka	Oct 17	4869.47	736603
12	Lusaka	Nov 17	4924.54	736634
13	Lusaka	Dec 17	4957.47	736664
14	Lusaka	Jan 18	5229.14	736695
15	Lusaka	Feb 18	5385.42	736726
16	Lusaka	Mar 18	5574.81	736754
17	Lusaka	Apr 18	5433.04	736785

Exploratory Data Analysis

[11]:

```
# Check number of data points
len(var_jctr_bnb_lusaka)

# Check data types
var_jctr_bnb_lusaka.info()

# Check statistics
var_jctr_bnb_lusaka.describe()
```

[11]:

```
18

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 4 columns):
Town      18 non-null object
Date      18 non-null object
```

```
Amount    18 non-null float64
DateX     18 non-null int64
dtypes: float64(1), int64(1), object(2)
memory usage: 656.0+ bytes
```

[11]:

	Amount	DateX
count	18.000000	18.000000
mean	5043.474444	736527.000000
std	212.033301	162.328643
min	4859.350000	736269.000000
25%	4925.497500	736396.750000
50%	4957.995000	736526.500000
75%	5014.102500	736656.500000
max	5574.810000	736785.000000

[12]:

```python
# Plot scatter plot
plt.scatter(var_jctr_bnb_lusaka["Date"], var_jctr_bnb_lusaka["Amount"])
plt.ylim(4500,5600)
plt.xlabel("BnB Date")
plt.ylabel("BnB Amount")
plt.xticks(rotation=90)
```

[12]:

```
<matplotlib.collections.PathCollection at 0x7f60c28e9780>
```

[12]:

```
(4500, 5600)
```

[12]:

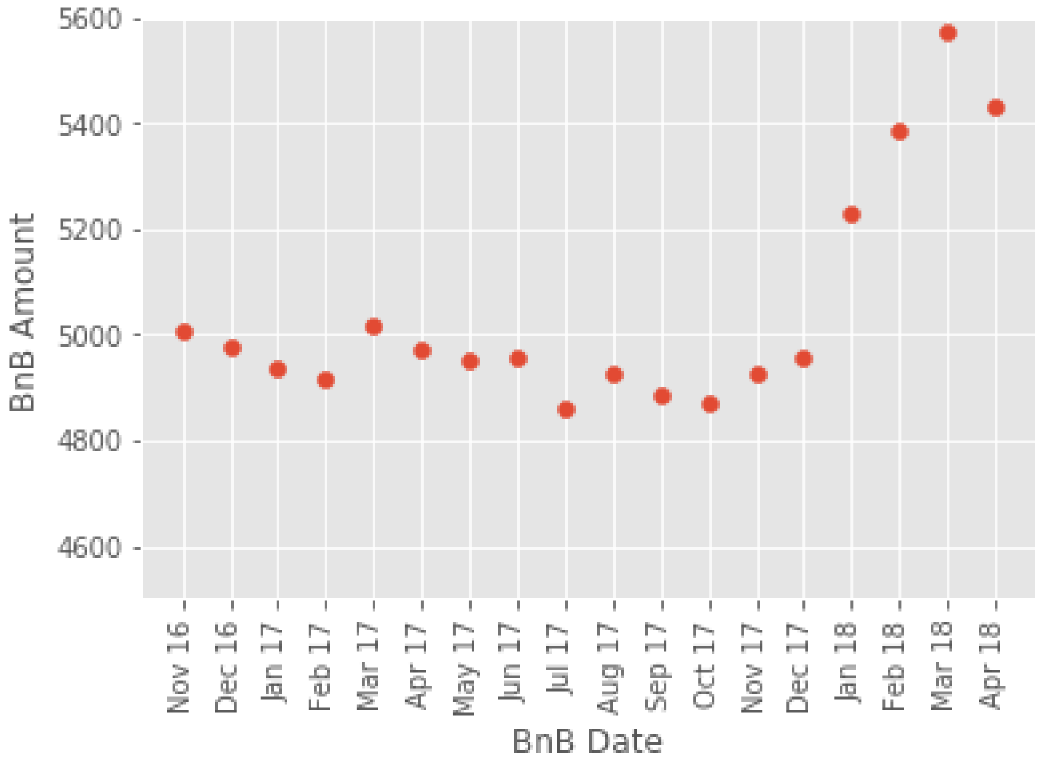
```
Text(0.5, 0, 'BnB Date')
```

[12]:

```
Text(0, 0.5, 'BnB Amount')
```

[12]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
 <a list of 18 Text xticklabel objects>)
```



Prepare Training and Testing Datasets

[13]:

```
###train_test_split?
var_x_train, var_x_test, var_y_train, var_y_test =␣
 ↪train_test_split(var_jctr_bnb_lusaka[["DateX"]], var_jctr_bnb_lusaka["Amount"],␣
 ↪test_size=0.10)
len(var_x_train)
len(var_x_test)
```

[13]:

```
16
```

[13]:

```
2
```

Applying Linear Regression

[14]:

```
# Create an instance of LinearRegression
var_jctr_lusaka_lg = LinearRegression()
```

[15]:

```
# Fit---Training
var_jctr_lusaka_lg.fit(var_x_train, var_y_train)
```

[15]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Evaluate Model Effectiveness

[16]:
```
import sklearn
var_jctr_lusaka_lg.predict(var_x_test)
print (var_y_test)

var_jctr_lusaka_lg.score(var_x_test, var_y_test)
r2_score(var_jctr_lusaka_lg.predict(var_x_test), var_y_test)
```

[16]:
```
array([5175.34182741, 4875.06045393])
```

```
14    5229.14
2     4935.46
Name: Amount, dtype: float64
```

[16]:
```
0.8482897479706637
```

[16]:
```
0.8548868112515664
```

Predicting Unknown Inputs

[17]:
```
# Predict sample values
# May 2018: 5,369.49
#
# March 2019: 5,543.16

var_may18 = datetime.strptime("May 18", "%b %y")
var_may18_input = var_may18.toordinal()
var_mar19 = datetime.strptime("May 19", "%b %y")
var_mar19_input = var_mar19.toordinal()

print ("++++++++++++++++++++++++++++++++++++")
print (var_may18)
var_jctr_lusaka_lg.predict([[var_may18_input]])
var_jctr_lusaka_lg.predict([[var_may18_input]])

print ("++++++++++++++++++++++++++++++++++++")
print (var_mar19)
var_jctr_lusaka_lg.predict([[var_mar19_input]])
```

```
++++++++++++++++++++++++++++++++++++
2018-05-01 00:00:00
```

[17]:
```
array([5274.06447075])
```

[17]:
```
array([5274.06447075])
```

```
++++++++++++++++++++++++++++++++++++
2019-05-01 00:00:00
```

[17]:
```
array([5574.34584424])
```

Visualising Results

[18]:

```
# Plot scatter plot
plt.scatter(var_jctr_bnb_lusaka["Date"], var_jctr_bnb_lusaka["Amount"], marker="^")
plt.ylim(4500,5600)
plt.title("JCTR BnB Predictions Using Linear Regression")
plt.xlabel("BnB Date")
plt.ylabel("BnB Amount")
plt.xticks(rotation=90)
plt.plot(var_jctr_bnb_lusaka["Date"], var_jctr_lusaka_lg.
 ↪predict(var_jctr_bnb_lusaka[["DateX"]]), color="red")
```

[18]:

```
<matplotlib.collections.PathCollection at 0x7f60bf034be0>
```

[18]:

```
(4500, 5600)
```

[18]:

```
Text(0.5, 1.0, 'JCTR BnB Predictions Using Linear Regression')
```

[18]:

```
Text(0.5, 0, 'BnB Date')
```

[18]:

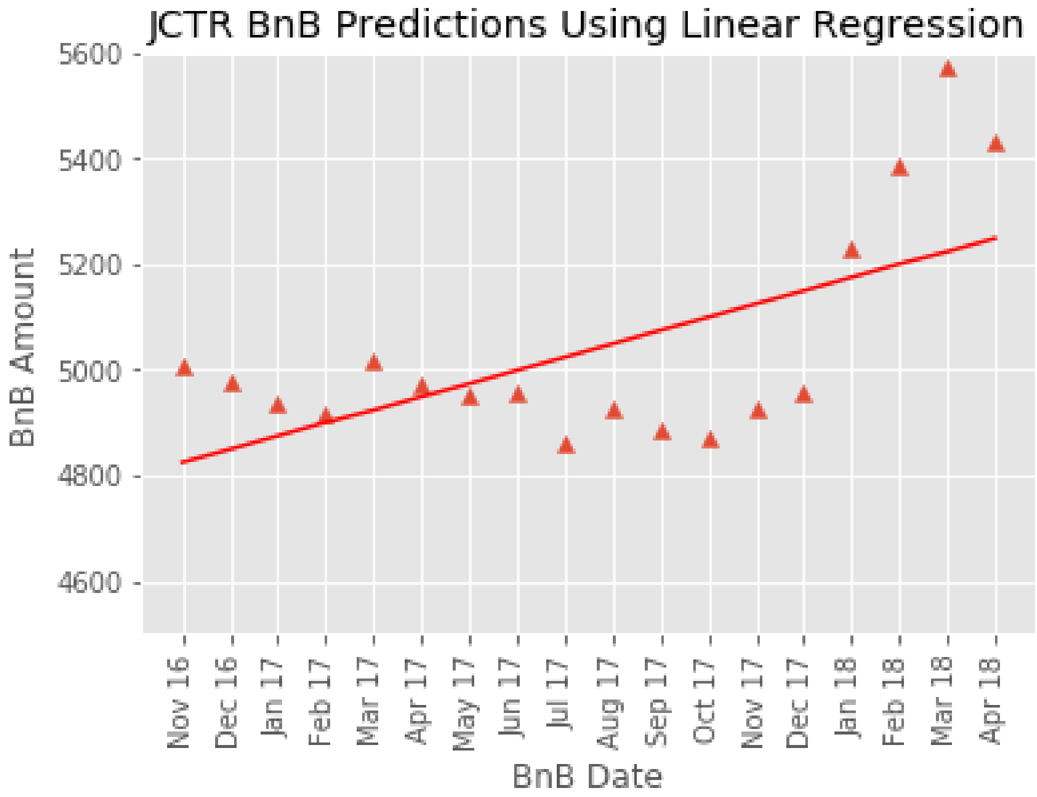
```
Text(0, 0.5, 'BnB Amount')
```

[18]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
 <a list of 18 Text xticklabel objects>)
```

[18]:

```
[<matplotlib.lines.Line2D at 0x7f60bf0235f8>]
```



Deriving Equation

[19]:

```python
# NOTE: Linking results to equation
# y = mx + c
var_jctr_lusaka_lg.coef_
var_jctr_lusaka_lg.intercept_
#
# f(x) = 0.80014412*x - 584288.4278808542
```

[19]:

```
array([0.82268869])
```

[19]:

```
-600895.3059570172
```

Exercise

1. Using the current model, compare the accuracy using an 80/20 train/test split

Multiple Linear Regression

Example #1: JCTR Zambian Towns BnB

Import Dataset Using Pandas

[20]:

```python
# Convert CSV dataset into pandas DataFrame
var_jctr_bnb_zambia = pd.read_csv("db-jctr-bnb-zambian_towns.csv", sep="|")
```

[21]:

```python
var_jctr_bnb_zambia.head(2)
```

[21]:

	Town	Date	Amount
0	Chinsali	Nov 16	2837.40
1	Chinsali	Dec 16	2788.35

Data Cleaning

[22]:

```python
# Checking for NULL values
var_jctr_bnb_zambia.isnull().values.any()
var_jctr_bnb_zambia[var_jctr_bnb_zambia.isna().any(axis=1)]

# Get aggregate means per town
var_jctr_bnb_zambia.groupby("Town").mean()

# Replace NaN entries using group aggregates
var_jctr_bnb_zambia["Amount"] = var_jctr_bnb_zambia.groupby("Town").transform(lambda␣
 ↪var_x: var_x.fillna(var_x.mean()))
```

[22]:

```
True
```

[22]:

	Town	Date	Amount
9	Chinsali	Aug 17	NaN
10	Chinsali	Sep 17	NaN
18	Chipata	Nov 16	NaN
53	Choma	Apr 18	NaN
71	Kabwe	Apr 18	NaN
77	Kasama	Apr 17	NaN
90	Kitwe	Nov 16	NaN
125	Livingstone	Apr 18	NaN
126	Luanshya	Nov 16	NaN
130	Luanshya	Mar 17	NaN
137	Luanshya	Oct 17	NaN
170	Mansa	Jul 17	NaN
171	Mansa	Aug 17	NaN
179	Mansa	Apr 18	NaN
185	Mongu	Apr 17	NaN
188	Mongu	Jul 17	NaN
209	Monze	Oct 17	NaN
224	Mpika	Jul 17	NaN
225	Mpika	Aug 17	NaN
257	Solwezi	Apr 17	NaN
263	Solwezi	Oct 17	NaN

[22]:

	Amount
Town	
Chinsali	2922.953750
Chipata	2563.340000
Choma	3720.444706
Kabwe	3501.190588
Kasama	3174.467647
Kitwe	3925.548235
Livingstone	3904.187647
Luanshya	3755.454000
Lusaka	5043.474444
Mansa	2971.382000
Mongu	3042.466875
Monze	3702.336471
Mpika	2939.607500
Ndola	4805.334444
Solwezi	4106.669375

[23]:
```
# Verify that NaN entries have been replaced
var_jctr_bnb_zambia[var_jctr_bnb_zambia.isna().any(axis=1)]
#
var_jctr_bnb_zambia[(var_jctr_bnb_zambia["Town"]=="Chinsali") &␣
 ↪(var_jctr_bnb_zambia["Date"]=="Aug 17")]
```

[23]:

Empty DataFrame Columns: Index(['Town', 'Date', 'Amount'], dtype='object') Index: Int64Index([], dtype='int64')

[23]:

	Town	Date	Amount
9	Chinsali	Aug 17	2922.95375

[24]:
```
# Convert date strings to date objects
var_jctr_bnb_zambia["Date"][0]
type(var_jctr_bnb_zambia["Date"][0])

# df['col'] = pd.to_datetime(df['col'])
# Please see documentation [1, 2] for details on working with date strings
# [1] http://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html
# [1] https://docs.python.org/3/library/datetime.html
###pd.to_datetime(var_jctr_bnb_zambia["Date"], format="%b %y").apply(lambda var_x:␣
 ↪var_x.toordinal())

# Replace date string with equivalent date objects
var_jctr_bnb_zambia["DateX"] = pd.to_datetime(var_jctr_bnb_zambia["Date"], format="%b␣
 ↪%y").apply(lambda var_x: var_x.toordinal())
```

[24]:
```
'Nov 16'
```

[24]:
```
str
```

[25]:
```
# Convert amount strings to numeric values
var_jctr_bnb_zambia["Amount"][0]
type(var_jctr_bnb_zambia["Amount"][0])
```

[25]:
```
2837.4
```

[25]:
```
numpy.float64
```

[26]:
```
var_jctr_bnb_zambia.head(2)
```

[26]:

	Town	Date	Amount	DateX
0	Chinsali	Nov 16	2837.40	736269
1	Chinsali	Dec 16	2788.35	736299

Exploratory Data Analysis

[27]:
```
# Check number of data points
len(var_jctr_bnb_zambia)

# Check data types
var_jctr_bnb_zambia.info()

# Check statistics
var_jctr_bnb_zambia.describe()
```

[27]:
```
270
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 4 columns):
Town      270 non-null object
Date      270 non-null object
Amount    270 non-null float64
DateX     270 non-null int64
dtypes: float64(1), int64(1), object(2)
memory usage: 8.5+ KB
```

[27]:

	Amount	DateX
count	270.000000	270.000000
mean	3605.257179	736527.000000
std	729.623036	158.048037
min	1398.400000	736269.000000
25%	2983.652500	736389.000000
50%	3647.505000	736526.500000
75%	4020.330000	736664.000000
max	5574.810000	736785.000000

[28]:

```
# Plot scatter plot
plt.scatter(var_jctr_bnb_zambia["Date"], var_jctr_bnb_zambia["Amount"])
plt.ylim(1300,5600)
plt.title("JCTR BnB Predictions Using Linear Regression")
plt.xlabel("BnB Date")
plt.ylabel("BnB Amount")
plt.xticks(rotation=90)
```

[28]:

```
<matplotlib.collections.PathCollection at 0x7f60bef638d0>
```

[28]:

```
(1300, 5600)
```

[28]:

```
Text(0.5, 1.0, 'JCTR BnB Predictions Using Linear Regression')
```

[28]:

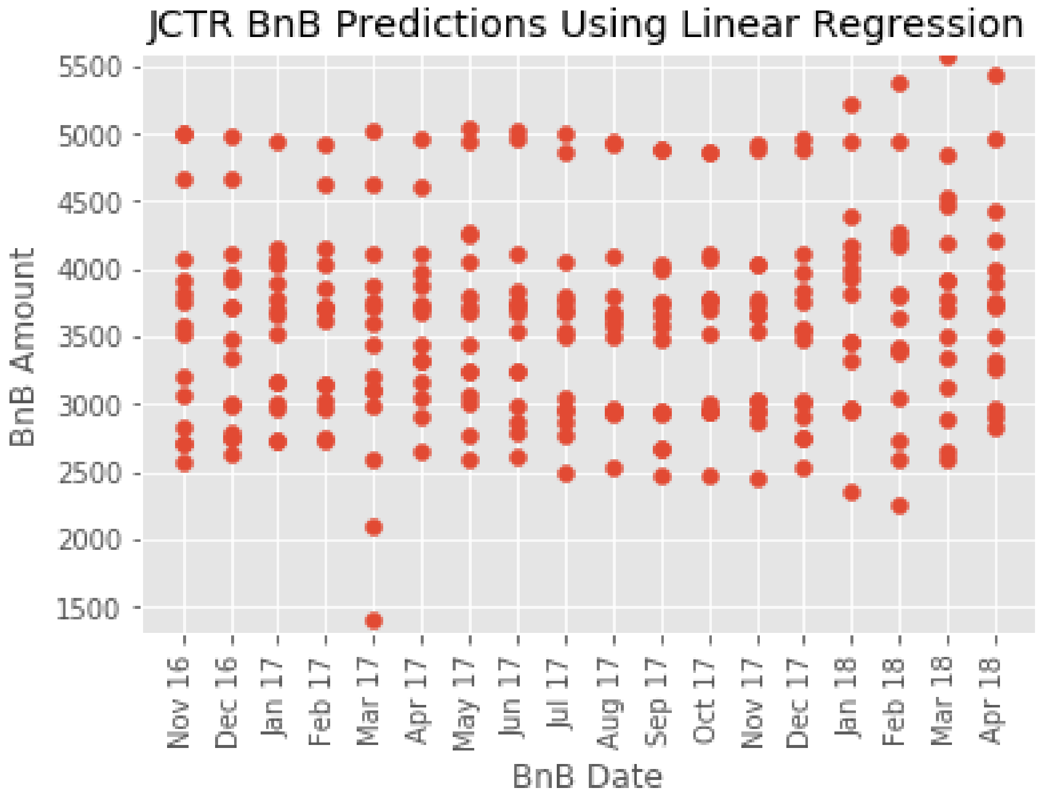
```
Text(0.5, 0, 'BnB Date')
```

[28]:

```
Text(0, 0.5, 'BnB Amount')
```

[28]:

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
 <a list of 18 Text xticklabel objects>)
```



[29]:

```
# Plot barplots
var_jctr_bnb_zambia.groupby("Town")["Amount"].mean().plot(kind="barh")
#plt.barh(var_jctr_bnb_zambia.groupby(["Town", "Date", "DateX"]).sum().reset_index())
plt.title("JCTR BnB Average Amounts")
plt.xlabel("BnB Amount")
plt.ylabel("Zambia Town")
plt.xticks(rotation=90)
```

[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f60bef23f98>
```

[29]:

```
Text(0.5, 1.0, 'JCTR BnB Average Amounts')
```

[29]:

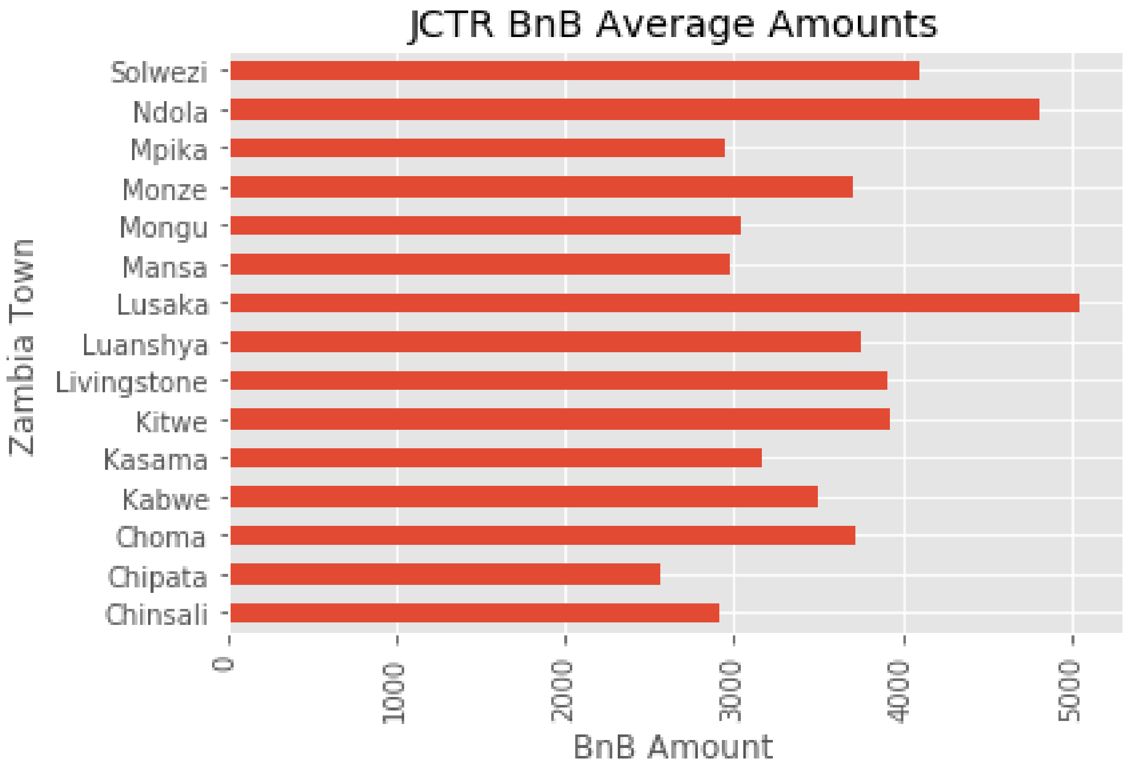
```
Text(0.5, 0, 'BnB Amount')
```

[29]:

```
Text(0, 0.5, 'Zambia Town')
```

[29]:

```
(array([   0., 1000., 2000., 3000., 4000., 5000., 6000.]),
 <a list of 7 Text xticklabel objects>)
```



On One Hot Encoding

- Town is textual and needs to be converted into a numeric format
- Assigning random numeric variables—e.g. Lusaka: 1, Chipata: 2, Mongu: 3, Livingstone: 5 simply does not work
- One Hot Encoding can be applied here
  - Create columns to each categories and assign binary values

[30]:
```
# Using pandas
pd.get_dummies(var_jctr_bnb_zambia["Town"]).head(2)
pd.get_dummies(var_jctr_bnb_zambia["Town"]).tail(2)
```

[30]:

	Chinsali	Chipata	Choma	Kabwe	Kasama	Kitwe	Livingstone	Luanshya	Lusaka	Mansa	Mongu
0	1	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	

[30]:

	Chinsali	Chipata	Choma	Kabwe	Kasama	Kitwe	Livingstone	Luanshya	Lusaka	Mansa	Mon
268	0	0	0	0	0	0	0	0	0	0	
269	0	0	0	0	0	0	0	0	0	0	

[31]:
```
# Using Scikit Learn
from sklearn.preprocessing import OneHotEncoder

var_ohe = OneHotEncoder(sparse=False)
```

```
var_ohe.fit_transform(var_jctr_bnb_zambia[["Town"]])[0]
```

[31]:
```
array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[32]:
```
# Using Scikit Learn
# For target vectors: e.g. ETD School Classification---You cannot use School Names are␣
 ↪targets
from sklearn.preprocessing import LabelEncoder

var_le = LabelEncoder()
var_le.fit_transform(var_jctr_bnb_zambia[["Town"]])
```

```
/home/lightonphiri/.local/lib/python3.6/site-packages/sklearn/preprocessing/label.py:235:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[32]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
        4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
        5,  5,  5,  5,  5,  5,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
        6,  6,  6,  6,  6,  6,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
        7,  7,  7,  7,  7,  7,  7,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
        8,  8,  8,  8,  8,  8,  8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,
        9,  9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10, 10, 10, 10, 10, 10,
       10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11,
       11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12,
       12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13, 13, 13,
       13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 14, 14, 14,
       14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14])
```

Reconstruct DataFrame with "Dummy" varibles

[33]:
```
# Use One Hot Encoding to Create Town column representation
var_town_dummies = pd.get_dummies(var_jctr_bnb_zambia["Town"])
var_town_dummies.head(2)
```

[33]:

	Chinsali	Chipata	Choma	Kabwe	Kasama	Kitwe	Livingstone	Luanshya	Lusaka	Mansa	Mongu
0	1	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	

[34]:
```
# Merge new DataFrame with "Dummy Variables" wit original DataFrame
var_jctr_bnb_zambia_input_ = pd.concat([var_jctr_bnb_zambia, var_town_dummies], axis=1)
var_jctr_bnb_zambia_input_.head(2)
```

[34]:

	Town	Date	Amount	DateX	Chinsali	Chipata	Choma	Kabwe	Kasama	Kitwe	Livingstone
0	Chinsali	Nov 16	2837.40	736269	1	0	0	0	0	0	0
1	Chinsali	Dec 16	2788.35	736299	1	0	0	0	0	0	0

[35]:

```
# Notes:
# var_jctr_bnb_zambia_input["Town"] can be dropped now
# var_jctr_bnb_zambia_input["Date"] can also be dropped
# To avoid "Dummy Variable Trap" drop one of the "Dummy Variables". If we drop␣
 ↪Solwezi, we can easily derive it
# Rule of Thumb: If you have N "Dummy Variables", drop one to have (N-1) "Dummy␣
 ↪Variables". Generally, "Dummy Variable Trap" is properly handled by␣
 ↪LinearRegression, but this is generally good practice

var_jctr_bnb_zambia_input = var_jctr_bnb_zambia_input_.drop(["Town", "Solwezi",␣
 ↪"Date"], axis=1)
var_jctr_bnb_zambia_input.head(2)
```

[35]:

	Amount	DateX	Chinsali	Chipata	Choma	Kabwe	Kasama	Kitwe	Livingstone	Luanshya	Lusal
0	2837.40	736269	1	0	0	0	0	0	0	0	
1	2788.35	736299	1	0	0	0	0	0	0	0	

Applying Linear Regression

[36]:

```
# Create an instance of LinearRegression
var_jctr_zambia_lg = LinearRegression()
```

Explicitly specify Independent Variables

- Dependent Variable(s): Amount
- Independent Variables (s): All but Amount

[37]:

```
# Independent Variables
var_jctr_bnb_zambia_input_X = var_jctr_bnb_zambia_input.drop("Amount", axis=1)
var_jctr_bnb_zambia_input_X.head(2)
```

[37]:

	DateX	Chinsali	Chipata	Choma	Kabwe	Kasama	Kitwe	Livingstone	Luanshya	Lusaka	Mansa
0	736269	1	0	0	0	0	0	0	0	0	0
1	736299	1	0	0	0	0	0	0	0	0	0

[38]:

```
# Dependent Variables
var_jctr_bnb_zambia_input_Y = var_jctr_bnb_zambia_input["Amount"]
var_jctr_bnb_zambia_input_Y.head(2)
```

[38]:

	Amount
0	2837.40

Continued on next page

	Amount
1	2788.35

Training the Machine Learning Model

[39]:
```
# Fit---Training the model
# Generally a time consuming process, when working with huge dataset
var_jctr_zambia_lg.fit(var_jctr_bnb_zambia_input_X, var_jctr_bnb_zambia_input_Y)
```

[39]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Predicting BnB Amounts

[40]:
```
# JCTR BnBs in March 2019
# Monze: ZMW 4,023.25
# 0,0,0,0,0,0,0,0,0,0,1,0,0,0
# Chipata: 3,512.99
# 0,1,0,0,0,0,0,0,0,0,0,0,0,0

var_mar19 = datetime.strptime("Mar 19", "%b %y")
var_mar19_input = var_mar19.toordinal()

print ("++++++++++++++++++++++++++++++++++++")
print (var_mar19)
var_jctr_zambia_lg.predict([[var_mar19_input,0,1,0,0,0,0,0,0,0,0,0,0,0,0]])
```

```
++++++++++++++++++++++++++++++++++++
2019-03-01 00:00:00
```

[40]:
```
array([2765.05074283])
```

Excercise #2

1. Using the model create above—modifying it if possible, predict the BnB amounts for Solwezi.
2. Modify the model by using BnB values for the past six(6) months and compare the accuracy with the current model.
3. Modify the model by incorporating the USD exchange rates and compare predictions with current model. Please use the Central Bank compiled intrest rates.

Visualising Results

[41]:
```
# Plot scatter plot
plt.scatter(var_jctr_bnb_zambia["Date"], var_jctr_bnb_zambia["Amount"], marker="^")
plt.ylim(1300,5600)
plt.title("JCTR BnB Predictions Using Linear Regression")
plt.xlabel("BnB Date")
plt.ylabel("BnB Amount")
plt.xticks(rotation=90)
var_jctr_bnb_zambia_input_X_chinsali =␣
 ↪var_jctr_bnb_zambia_input_X[var_jctr_bnb_zambia_input_X["Chinsali"]==1]
```

```
var_jctr_bnb_zambia_input_Y_chinsali =␣
 ↪var_jctr_bnb_zambia_input_[var_jctr_bnb_zambia_input_["Chinsali"]==1]["Date"]

plt.plot(var_jctr_bnb_zambia_input_Y_chinsali, var_jctr_zambia_lg.
 ↪predict(var_jctr_bnb_zambia_input_X_chinsali), color="red")
```

[41]:
```
<matplotlib.collections.PathCollection at 0x7f60bee32e48>
```

[41]:
```
(1300, 5600)
```

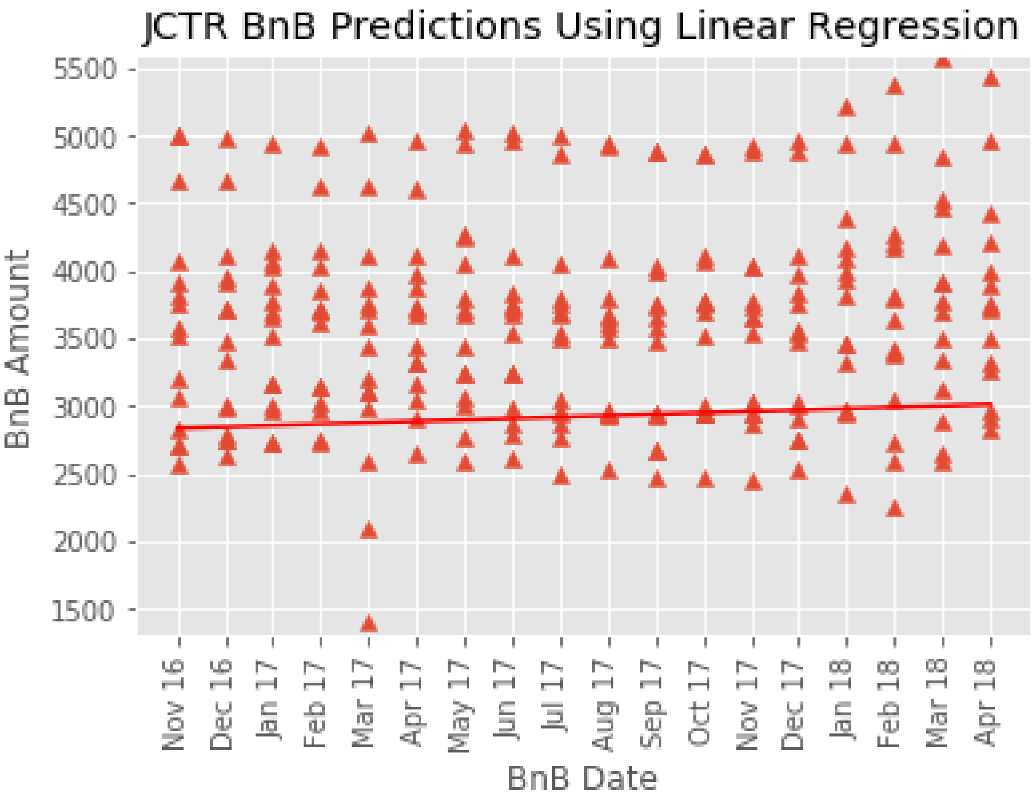
[41]:
```
Text(0.5, 1.0, 'JCTR BnB Predictions Using Linear Regression')
```

[41]:
```
Text(0.5, 0, 'BnB Date')
```

[41]:
```
Text(0, 0.5, 'BnB Amount')
```

[41]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17],
 <a list of 18 Text xticklabel objects>)
```

[41]:
```
[<matplotlib.lines.Line2D at 0x7f60bee22828>]
```



Deriving Equation

[42]:
```
# NOTE: Linking results to equation
# y = mx + c
var_jctr_zambia_lg.coef_
var_jctr_zambia_lg.intercept_
#
# f(x) = 0.34072761*x1 + -1183.715625*x2 [...] + -246848.41218698895
```

[42]:
```
array([    0.34072761, -1183.715625  , -1543.329375  ,  -386.22466912,
        -605.47878676,  -932.20172794,  -181.12113971,  -202.48172794,
        -351.215375  ,   936.80506944, -1135.287375  , -1064.2025    ,
        -404.33290441, -1167.061875  ,   698.66506944])
```

[42]:
```
-246848.41218698895
```

Single Variable Usnig K Fold Cross-Validation

[43]:
```
# 1. DataFrame
var_jctr_bnb_lusaka.head(2)
var_jctr_bnb_lusaka_kfold_X = var_jctr_bnb_lusaka["DateX"]
var_jctr_bnb_lusaka_kfold_Y = var_jctr_bnb_lusaka["Amount"]
```

[43]:

	Town	Date	Amount	DateX
0	Lusaka	Nov 16	5005.14	736269
1	Lusaka	Dec 16	4976.67	736299

[44]:
```
# 2. Create 3 folds
var_kfold = KFold(n_splits=3)
```

[45]:
```
# 3. Loop through folds and train + test model
for var_train_index, var_test_index in var_kfold.split(var_jctr_bnb_lusaka_kfold_X):
    var_x_train_k, var_x_test_k, var_y_train_k, var_y_test_k =␣
 ↪var_jctr_bnb_lusaka_kfold_X[var_train_index],␣
 ↪var_jctr_bnb_lusaka_kfold_X[var_test_index],␣
 ↪var_jctr_bnb_lusaka_kfold_Y[var_train_index],␣
 ↪var_jctr_bnb_lusaka_kfold_Y[var_test_index]

len(var_x_train_k)
len(var_y_train_k)

len(var_x_test_k)
len(var_y_test_k)
```

[45]:
```
12
```

[45]:
```
12
```

[45]:
```
6
```

[45]:
```
6
```

[ ]:
```

```