In [0]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas import DataFrame, Series
from time import strftime
import seaborn as sns
import datetime
import sqlite3
```

In [0]:

```
#Read in .csv (trag the csv format dataset into local file, follow
shots = pd.read_csv("shot_logs.csv")
playoffs = pd.read_csv("nba_playoff_teams_2015.csv")
```

In [0]:

```
#from google.colab import drive
#drive.mount('/content/drive')
```

In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

In [0]:

```
#Read in .csv using google drive
#shots = pd.read_csv("/content/drive/My Drive/shot_logs.csv")
#playoffs = pd.read_csv("/content/drive/My Drive/nba_playoff_teams_
```

In [3]:

```
#Display top 5 rows of shots
shots.head()
```

Out[3]:

	GAME_ID	MATCHUP	LOCATION	W	FINAL_MARGIN	SHOT_NUMBER
0	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	1
1	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	2
2	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	3
3	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	4
4	21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	5

In [4]:

```
#Display shots dimensions
shots.shape
```

Out[4]:

```
(128069, 21)
```

In [0]:

```
shots.drop(['LOCATION','FINAL_MARGIN', 'SHOT_RESULT',
            'CLOSEST_DEFENDER','CLOSEST_DEFENDER_PLAYER_ID',
            'PTS'], axis = 1, inplace = True)
```

In [6]:

```
shots.head()
```

Out[6]:

	GAME_ID	MATCHUP	W	SHOT_NUMBER	PERIOD	GAME_CLOCK	SH
0	21400899	MAR 04, 2015 - CHA @ BKN	W	1	1	1:09	
1	21400899	MAR 04, 2015 - CHA @ BKN	W	2	1	0:14	
2	21400899	MAR 04, 2015 - CHA @ BKN	W	3	1	0:00	
3	21400899	MAR 04, 2015 - CHA @ BKN	W	4	2	11:47	
4	21400899	MAR 04, 2015 - CHA @ BKN	W	5	2	10:34	

In [7]:

```
#List of all variables and data types of shots
shots.dtypes
```

Out[7]:

```
GAME_ID                int64
MATCHUP               object
W                     object
SHOT_NUMBER            int64
PERIOD                 int64
GAME_CLOCK            object
SHOT_CLOCK           float64
DRIBBLES               int64
TOUCH_TIME           float64
SHOT_DIST            float64
PTS_TYPE               int64
CLOSE_DEF_DIST       float64
FGM                    int64
player_name           object
player_id              int64
dtype: object
```

In [8]:

```
#Descriptive statistics of shot data numeric variables
shots.describe()
```

Out[8]:

	GAME_ID	SHOT_NUMBER	PERIOD	SHOT_CLOCK	D
count	1.280690e+05	128069.000000	128069.000000	122502.000000	12806
mean	2.140045e+07	6.506899	2.469427	12.453344	
std	2.578773e+02	4.713260	1.139919	5.763265	
min	2.140000e+07	1.000000	1.000000	0.000000	
25%	2.140023e+07	3.000000	1.000000	8.200000	
50%	2.140045e+07	5.000000	2.000000	12.300000	
75%	2.140067e+07	9.000000	3.000000	16.675000	
max	2.140091e+07	38.000000	7.000000	24.000000	3

In [9]:

```
#Check for missing data
shots.isnull().sum()
```

Out[9]:

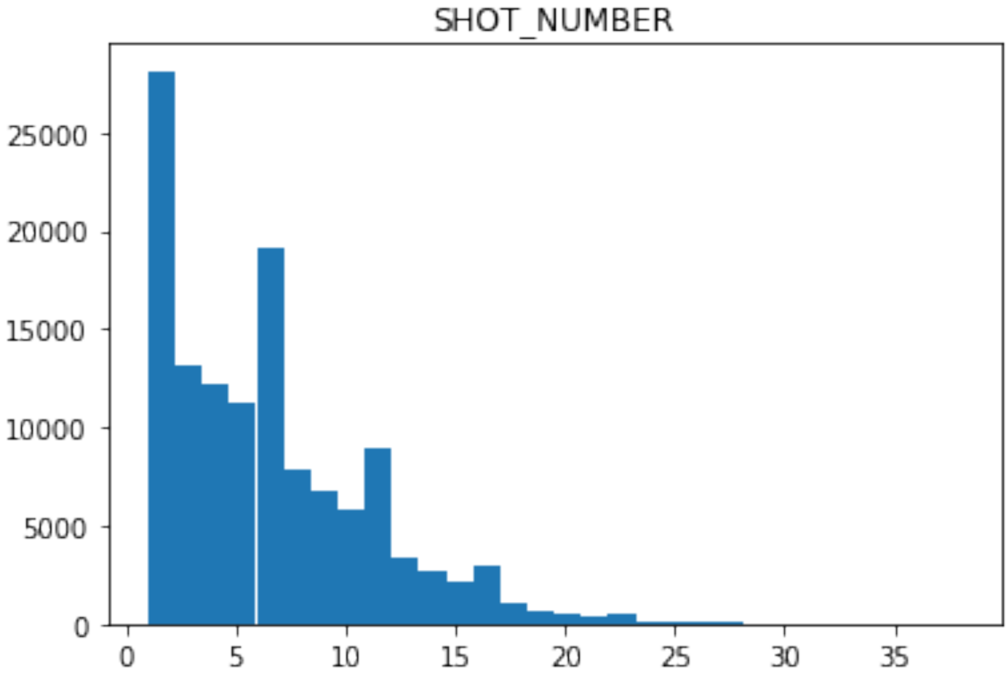
```
GAME_ID               0
MATCHUP               0
W                     0
SHOT_NUMBER           0
PERIOD                0
GAME_CLOCK            0
SHOT_CLOCK         5567
DRIBBLES              0
TOUCH_TIME            0
SHOT_DIST             0
PTS_TYPE              0
CLOSE_DEF_DIST        0
FGM                   0
player_name           0
player_id             0
dtype: int64
```

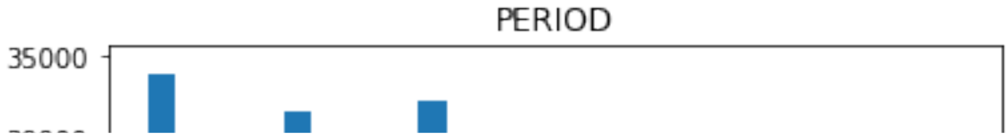
In [0]:

```
#Create a list of numeric variables to plot in order to see the dis
num_var_list = ['SHOT_NUMBER','PERIOD','SHOT_CLOCK','DRIBBLES','TOU
```

In [11]:

```
for column in num_var_list:
  plt.hist(shots[column], bins=30)
  plt.title(column)
  plt.show()
```


SHOT_NUMBER


PERIOD

In [12]:

```
#Removing rows with touch time < 0
shots = shots[shots.TOUCH_TIME > 0]
shots['TOUCH_TIME'].describe()
```

Out[12]:

```
count    124711.000000
mean          2.846596
std           2.991336
min           0.100000
25%           0.900000
50%           1.700000
75%           3.800000
max          24.900000
Name: TOUCH_TIME, dtype: float64
```

In [13]:

```
#Editing shot clock values that are missing
shots['SHOT_CLOCK'] = shots['SHOT_CLOCK'].fillna(shots['GAME_CLOCK'
#Check for missing data
shots.isnull().sum()
```

Out[13]:

```
GAME_ID          0
MATCHUP          0
W                0
SHOT_NUMBER      0
PERIOD           0
GAME_CLOCK       0
SHOT_CLOCK       0
DRIBBLES         0
TOUCH_TIME       0
SHOT_DIST        0
PTS_TYPE         0
CLOSE_DEF_DIST   0
FGM              0
player_name      0
player_id        0
dtype: int64
```

In [0]:

```
#Convert "GAME_CLOCK" variable to minute:second time data type
shots['GAME_CLOCK']=pd.to_datetime(shots['GAME_CLOCK'],format='%m:%
```

In [0]:

```
#To create a variable for the team the player is on, we need to spl
#and remove the opposing team.  By splitting and removing whitespac
#in an SQL join to link the shots table with a table providing info
shots[['Date', 'Teams']] = shots['MATCHUP'].str.split('-', 1, expan
shots.drop(['MATCHUP', 'Date'], axis = 1, inplace = True)
shots['Teams'] = shots['Teams'].str.strip()
shots[['Team', 'Opposing_Team']] = shots['Teams'].str.split(' ', 1,
shots.drop(['Teams', 'Opposing_Team'], axis = 1, inplace = True)
shots['Team'] = shots['Team'].str.strip()
```

In [16]:

```python
shots.head()
```

Out[16]:

	GAME_ID	W	SHOT_NUMBER	PERIOD	GAME_CLOCK	SHOT_CLOCK
0	21400899	W	1	1	1:09	10.8
1	21400899	W	2	1	0:14	3.4
2	21400899	W	3	1	0:00	0:00
3	21400899	W	4	2	11:47	10.3
4	21400899	W	5	2	10:34	10.9

In [17]:

```python
#View playoffs df
playoffs
```

Out[17]:

	team	made_playoffs
0	ATL	Yes
1	BKN	Yes
2	BOS	Yes
3	CHA	No
4	CHI	Yes
5	CLE	Yes
6	DAL	Yes
7	DEN	No
8	DET	No
9	GSW	Yes
10	HOU	Yes
11	IND	No
12	LAC	Yes

13	LAL	No
14	MEM	Yes
15	MIA	No
16	MIL	Yes
17	MIN	No
18	NOP	Yes
19	NYK	No
20	OKC	No
21	ORL	No
22	PHI	No
23	PHX	No
24	POR	Yes
25	SAC	No
26	SAS	Yes
27	TOR	Yes
28	UTA	No
29	WAS	Yes

Create two SQL tables to store the data

In [18]:

```
#Open database connection
conn = sqlite3.connect("nba_shot_data_2015.db")
print(conn)
```

```
<sqlite3.Connection object at 0x7f4e1a3ce730>
```

In [19]:

```
conn.execute("DROP TABLE IF EXISTS `Shots_Table`")
print("Table dropped")
```

```
Table dropped
```

In [20]:

```
shots.dtypes
```

Out[20]:

```
GAME_ID             int64
W                  object
SHOT_NUMBER         int64
PERIOD              int64
GAME_CLOCK         object
SHOT_CLOCK         object
DRIBBLES            int64
TOUCH_TIME        float64
SHOT_DIST         float64
PTS_TYPE            int64
CLOSE_DEF_DIST    float64
FGM                 int64
player_name        object
player_id           int64
Team               object
dtype: object
```

In [21]:

```
#Create an SQL table to store shots data
try:
    conn.execute('''CREATE TABLE Shots_Table
       (GAME_ID                                  INTEGER  NOT NULL,
        W                                        TEXT,
        SHOT_NUMBER                              INTEGER  NOT NULL,
        PERIOD                                   INTEGER,
        GAME_CLOCK                               TEXT,
        SHOT_CLOCK                               FLOAT,
        DRIBBLES                                 INTEGER,
        TOUCH_TIME                               FLOAT,
        SHOT_DIST                                FLOAT,
        PTS_TYPE                                 INTEGER,
        CLOSE_DEF_DIST                           FLOAT,
        FGM                                      INTEGER,
        PLAYER_NAME                              TEXT,
        PLAYER_ID                                INTEGER  NOT NULL,
        TEAM                                     TEXT,
        PRIMARY KEY (GAME_ID, PLAYER_ID, SHOT_NUMBER));''')
    print("Table created successfully")
except Exception as e:
    print(str(e))
    print('Table creation failed!')
finally:
    conn.close()
```

```
Table created successfully
```

In [0]:

```
#null_counts = shots.isnull().sum()
#null_counts[null_counts > 0].sort_values(ascending=False)
```

In [23]:

```
#Create a list of variables from shots df and insert into sql table
shots_list = shots.values.tolist()

conn = sqlite3.connect("nba_shot_data_2015.db")

cursor = conn.cursor()

try:
    cursor.executemany('''
      INSERT INTO Shots_Table (
          GAME_ID,
          W,
          SHOT_NUMBER,
          PERIOD,
          GAME_CLOCK,
          SHOT_CLOCK,
          DRIBBLES,
          TOUCH_TIME,
          SHOT_DIST,
          PTS_TYPE,
          CLOSE_DEF_DIST,
          FGM,
          PLAYER_NAME,
          PLAYER_ID,
          TEAM)
      VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)''', shots_list)
    conn.commit()
    print("Data Inserted successfully")
except Exception as e:
    print(str(e))
    print("Insertion failed!")
finally:
    conn.close()
```

```
Data Inserted successfully
```

In [24]:

```
#Print the number of rows in the Shots_Table
conn = sqlite3.connect("nba_shot_data_2015.db")
cursor = conn.cursor()
cursor.execute("SELECT count(*) FROM Shots_Table;")
rows = cursor.fetchall()
for row in rows:
    print(row)
conn.close()
```

```
(124711,)
```

In [25]:

```
#Open database connection
conn = sqlite3.connect("nba_shot_data_2015.db")
print(conn)
```

```
<sqlite3.Connection object at 0x7f4e1a3ce730>
```

In [26]:

```
conn.execute("DROP TABLE IF EXISTS `Playoffs_Table`")
print("Table dropped")
```

```
Table dropped
```

In [27]:

```
playoffs.dtypes
```

Out[27]:

```
team                object
made_playoffs       object
dtype: object
```

In [28]:

```
#Create an SQL table to store playoff team data
try:
    conn.execute('''CREATE TABLE Playoffs_Table
       (TEAM                                    TEXT  PRIMARY KEY,
        PLAYOFFS                                TEXT);''')
    print("Table created successfully")
except Exception as e:
    print(str(e))
    print('Table creation failed!')
finally:
    conn.close()
```

```
Table created successfully
```

In [29]:

```
#Create a list of variables from playoffs df and insert into sql ta
playoffs_list = playoffs.values.tolist()

conn = sqlite3.connect("nba_shot_data_2015.db")

cursor = conn.cursor()

try:
    cursor.executemany('''
      INSERT INTO Playoffs_Table (
          TEAM,
          PLAYOFFS)
      VALUES (?,?)''', playoffs_list)
    conn.commit()
    print("Data Inserted successfully")
except Exception as e:
    print(str(e))
    print("Insertion failed!")
finally:
    conn.close()
```

```
Data Inserted successfully
```

In [30]:

```
#Print the number of rows in the Playoffs_Table
conn = sqlite3.connect("nba_shot_data_2015.db")
cursor = conn.cursor()
cursor.execute("SELECT count(*) FROM Playoffs_Table;")
rows = cursor.fetchall()
for row in rows:
    print(row)
conn.close()
```

```
(30,)
```

In [31]:

```
conn = sqlite3.connect("nba_shot_data_2015.db")
print(conn)
conn.execute("DROP TABLE IF EXISTS `combined_df`")
print("Table dropped")
```

```
<sqlite3.Connection object at 0x7f4dea0651f0>
Table dropped
```

In [32]:

```
conn = sqlite3.connect("nba_shot_data_2015.db")

try:
  combined_df = pd.read_sql_query('''
        CREATE TABLE combined_df as
        SELECT *
        FROM Shots_Table as a
        JOIN Playoffs_Table as b
        ON a.TEAM=b.TEAM;''', conn)
except Exception as e:
  print(str(e))
finally:
  conn.close()
```

```
'NoneType' object is not iterable
```

In [33]:

```
#Print the number of rows in the combined_df table
conn = sqlite3.connect("nba_shot_data_2015.db")
cursor = conn.cursor()
cursor.execute("SELECT count(*) FROM combined_df;")
rows = cursor.fetchall()
for row in rows:
    print(row)
conn.close()
```

```
(124711,)
```

In [34]:

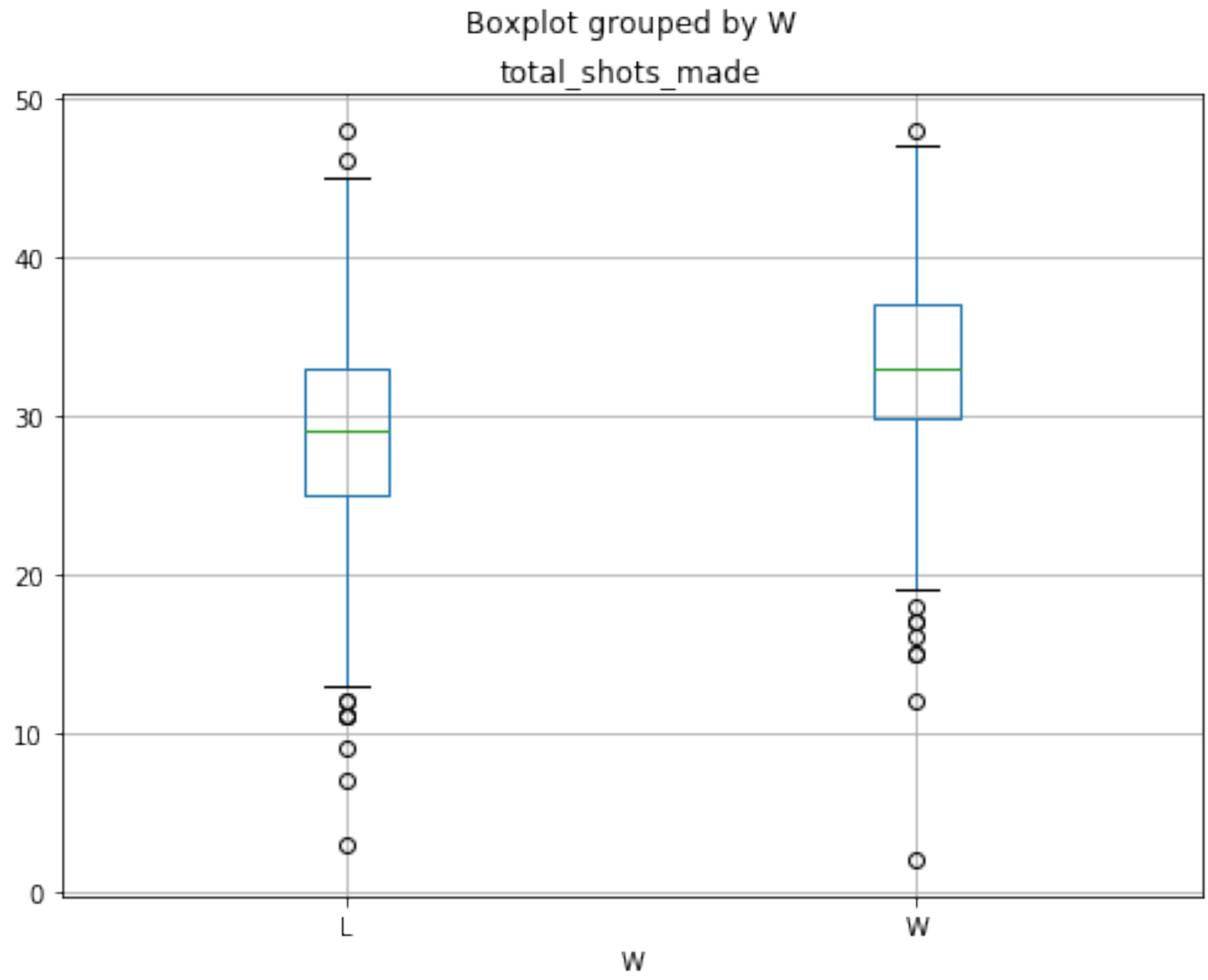
```
#To find the average number of shots made per game based on win or
conn = sqlite3.connect("nba_shot_data_2015.db")
try:
  df = pd.read_sql_query('''
      SELECT GAME_ID, W, SUM(FGM) AS total_shots_made
      FROM combined_df
      GROUP BY GAME_ID, W''', conn)
except Exception as e:
  print(str(e))
finally:
  conn.close()
df.head()
```

Out[34]:

	GAME_ID	W	total_shots_made
0	21400001	L	27
1	21400001	W	34
2	21400002	L	31
3	21400002	W	37
4	21400003	L	25

In [35]:

```
# Make a boxplot graph using pandas
df.boxplot(column = 'total_shots_made', by = 'W', figsize = (8,6))
plt.show()
```



The average number of shots made in a win was 33.2 and average shots made in a loss was 29.1.

In [36]:

```
# To find the overall field goal percentage of teams making the pla
conn = sqlite3.connect("nba_shot_data_2015.db")
try:
  df = pd.read_sql_query('''
  SELECT PLAYOFFS, avg(FGM) AS fg_percentage
  FROM combined_df
  GROUP BY PLAYOFFS''', conn)
except Exception as e:
  print(str(e))
finally:
  conn.close()
df.head()
```

Out[36]:

	PLAYOFFS	fg_percentage
0	No	0.442278
1	Yes	0.459980

The overall field goal percentage of teams making the playoffs was 46.0% compared to the field goal percentage of teams not making the playoffs which was 44.2%.

In [37]:

```
# To find the top 3 players with the highest 3 point shooting perce
conn = sqlite3.connect("nba_shot_data_2015.db")
try:
  df = pd.read_sql_query('''
  SELECT PLAYER_ID, PLAYER_NAME, TEAM, count(PLAYER_ID), AVG(FGM) A
  FROM combined_df
  WHERE PTS_TYPE = 3
  GROUP BY PLAYER_ID
  HAVING COUNT(PLAYER_ID) >= 10
  ORDER BY AVERAGE_FGM DESC ''', conn)
except Exception as e:
  print(str(e))
finally:
  conn.close()
df.head()
```

Out[37]:

	PLAYER_ID	PLAYER_NAME	TEAM	count(PLAYER_ID)	AVERAGE_FGM
0	202337	luke babbitt	NOP	102	0.509804
1	2594	kyle korver	ATL	354	0.497175
2	2225	tony parker	SAS	68	0.470588
3	203932	aaron gordon	ORL	20	0.450000
4	202087	alonzo gee	DEN	25	0.440000

The 3 players with the highest 3 point shooting percentage among players that attempted at least ten 3 point shots were Luke Babbit (NOP/51.0%), Kyle Korver (ATL/49.7%) and Tony Parker (SAS/47.1%).

In [38]:

```
# The average number of dribbles when player missed or made the sho
conn = sqlite3.connect("nba_shot_data_2015.db")
try:
  df = pd.read_sql_query('''
  SELECT AVG(DRIBBLES) AS average_dribbles,FGM
  FROM combined_df
  GROUP BY FGM''', conn)
except Exception as e:
  print(str(e))
finally:
  conn.close()
df.head()
```

Out[38]:

	average_dribbles	FGM
0	2.188235	0
1	1.943661	1

In [39]:

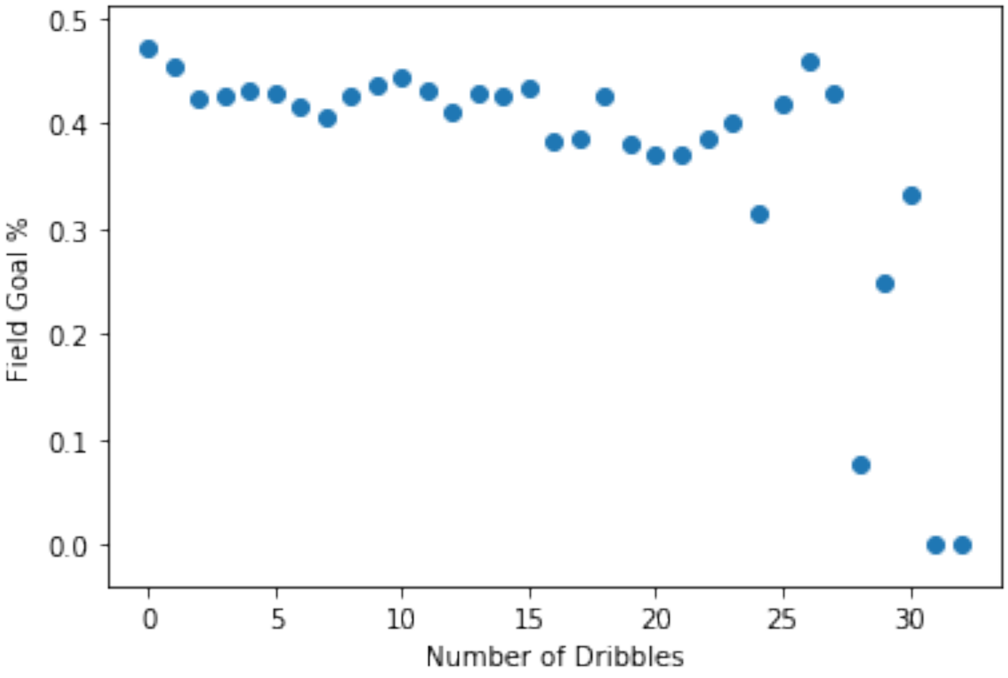
```
# The average number of dribbles when player missed or made the sho
conn = sqlite3.connect("nba_shot_data_2015.db")
try:
  df = pd.read_sql_query('''
  SELECT DRIBBLES,
  AVG(FGM) as FG_Pct
  FROM combined_df
  GROUP BY DRIBBLES''', conn)
except Exception as e:
  print(str(e))
finally:
  conn.close()
df.head()
```

Out[39]:

	DRIBBLES	FG_Pct
0	0	0.472363
1	1	0.454049
2	2	0.424520
3	3	0.425857
4	4	0.431429

In [40]:

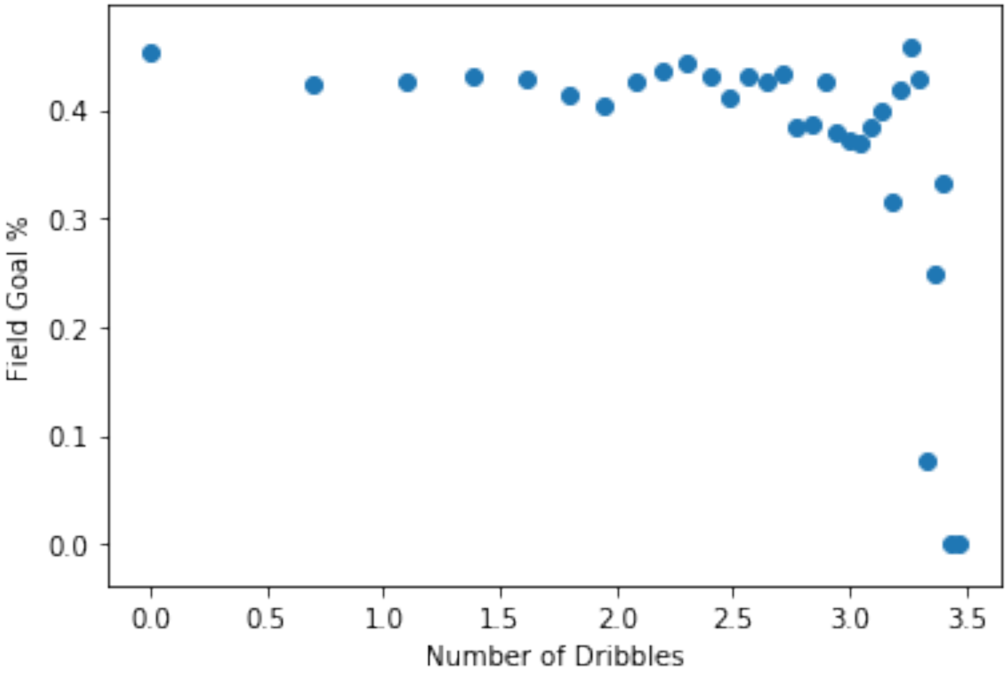
```
plt.scatter(df['DRIBBLES'],df['FG_Pct'])
plt.xlabel("Number of Dribbles")
plt.ylabel("Field Goal %")
plt.show()
```



In [41]:

```
#revierd scatter plot using lognormal transformation because number
plt.scatter(np.log(df.DRIBBLES),df['FG_Pct'])
plt.xlabel("Number of Dribbles")
plt.ylabel("Field Goal %")
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/ser
ies.py:856: RuntimeWarning: divide by zero encountered
in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



On average, players took about 0.25 more dribbles when they missed the shot than when they made the shot.

However when reviewing the scatter plot there does not appear to be any correlation between the number of dribbles and the Field Goal %

In [42]:

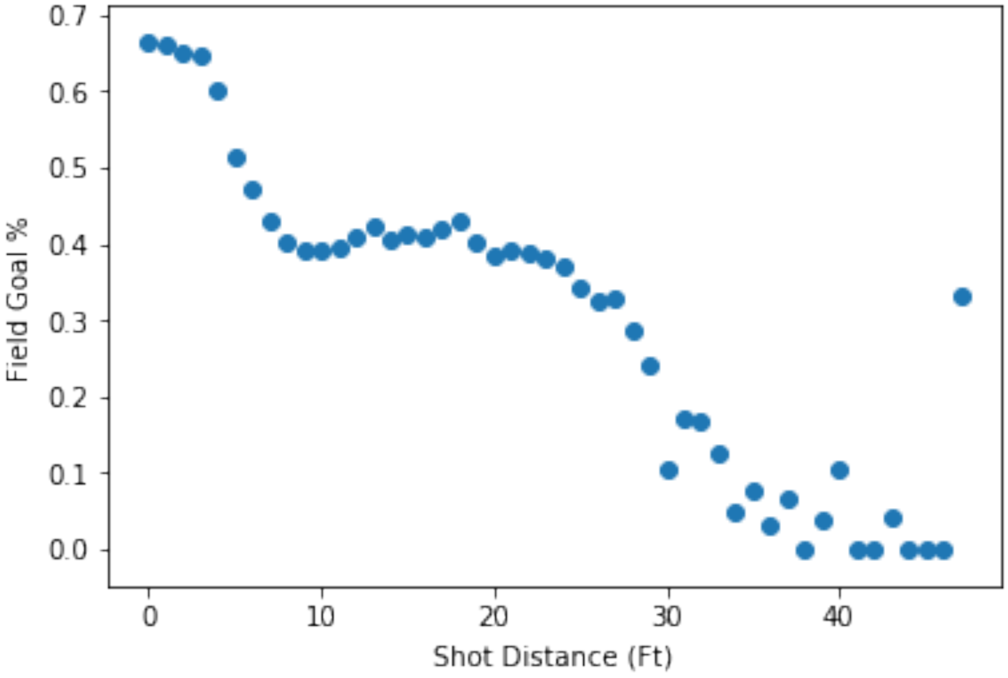
```
#compare distance to % FG Made
conn = sqlite3.connect("nba_shot_data_2015.db")
try:
  df = pd.read_sql_query('''
  SELECT ROUND(SHOT_DIST,0) AS Shot_Dist_Ft,
  AVG(FGM) as FG_pct
  FROM combined_df
  GROUP BY ROUND(SHOT_DIST,0)''', conn)
except Exception as e:
  print(str(e))
finally:
  conn.close()
df.head()
```

Out[42]:

	Shot_Dist_Ft	FG_pct
0	0.0	0.663918
1	1.0	0.660341
2	2.0	0.649905
3	3.0	0.647447
4	4.0	0.602420

In [43]:

```
plt.scatter(df['Shot_Dist_Ft'],df['FG_pct'])
plt.xlabel("Shot Distance (Ft)")
plt.ylabel("Field Goal %")
plt.show()
```



In [0]:

```
#as expected shot, field goal % decreases as shot distance increase
#that there seems to be little difference between 10ft and 20ft.
```

In [45]:

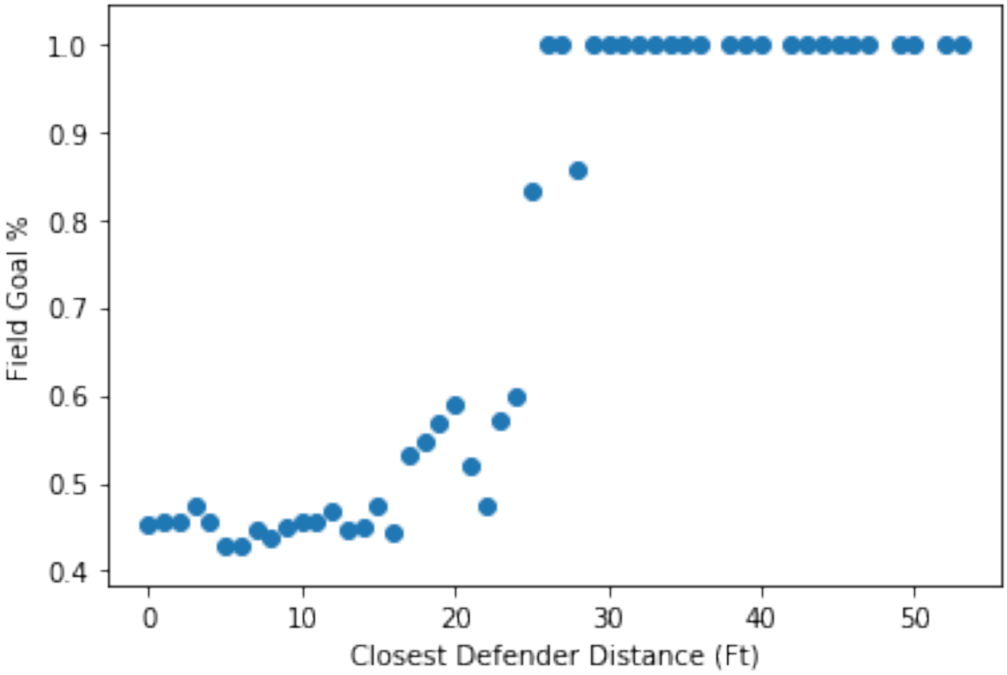
```
#compare distance of defender to % FG Made
conn = sqlite3.connect("nba_shot_data_2015.db")
try:
  df = pd.read_sql_query('''
  SELECT ROUND(CLOSE_DEF_DIST,0) AS Closest_Def,
  AVG(FGM) as FG_pct
  FROM combined_df
  GROUP BY ROUND(CLOSE_DEF_DIST,0)''', conn)
except Exception as e:
  print(str(e))
finally:
  conn.close()
df.head()
```

Out[45]:

	Closest_Def	FG_pct
0	0.0	0.453280
1	1.0	0.455824
2	2.0	0.455368
3	3.0	0.473128
4	4.0	0.454989

In [46]:

```
plt.scatter(df['Closest_Def'],df['FG_pct'])
plt.xlabel("Closest Defender Distance (Ft)")
plt.ylabel("Field Goal %")
plt.show()
```



As expected, field goal % increases as there is a greater distance between the shooter and the defender. At distances greater than 25ft the shot is almost 100%.

In [47]:

```
#Response Variable: FGM
#Predicting Variables: shot_number,dribbles,touch_time,shot_distanc
#The new dataset obtains features of interests.

shots_new=DataFrame(shots,columns=['SHOT_NUMBER','DRIBBLES','TOUCH_
shots_new.to_csv(r'sample_data/shot_new.csv')
shots_new.head()
```

Out[47]:

	SHOT_NUMBER	DRIBBLES	TOUCH_TIME	SHOT_DIST	CLOSE_DEF_DIS
0	1	2	1.9	7.7	1
1	2	0	0.8	28.2	6
2	3	3	2.7	10.1	0
3	4	2	1.9	17.2	3
4	5	2	2.7	3.7	1

In [0]:

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-2.4.4/spark-2.4
!tar xf spark-2.4.4-bin-hadoop2.7.tgz
!pip install -q findspark
```

In [0]:

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.4-bin-hadoop2.7"
```

In [50]:

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark=SparkSession.builder.master("local[*]").getOrCreate()
print(spark)
```

```
<pyspark.sql.session.SparkSession object at 0x7f4de7be
3978>
```

In [51]:

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
dataset=spark.read.csv('sample_data/shot_new.csv',inferSchema=True,
dataset.show()
dataset.printSchema()
```

```
+---+-----------+--------+----------+---------+-------
-------+---+
|_c0|SHOT_NUMBER|DRIBBLES|TOUCH_TIME|SHOT_DIST|CLOSE_D
EF_DIST|FGM|
+---+-----------+--------+----------+---------+-------
-------+---+
|  0|          1|       2|       1.9|      7.7|
1.3|  1|
|  1|          2|       0|       0.8|     28.2|
6.1|  0|
|  2|          3|       3|       2.7|     10.1|
0.9|  0|
|  3|          4|       2|       1.9|     17.2|
3.4|  0|
|  4|          5|       2|       2.7|      3.7|
1.1|  0|
|  5|          6|       2|       4.4|     18.4|
2.6|  0|
|  6|          7|      11|       9.0|     20.7|
6.1|  0|
|  7|          8|       3|       2.5|      3.5|
2.1|  1|
|  8|          9|       0|       0.8|     24.6|
7.3|  0|
|  9|          1|       0|       1.1|     22.4|
19.8|  0|
| 10|          2|       8|        7.5|     24.5|
4.7|  0|
| 11|          3|      14|       11.9|     14.6|
1.8|  1|
| 12|          4|       2|        2.9|      5.9|
5.4|  1|
| 13|          1|       0|        0.8|     26.4|
4.4|  0|
| 14|          1|       0|        0.5|     22.8|
5.3|  0|
| 15|          2|       3|        2.7|     24.7|
5.6|  1|
| 16|          3|       6|        5.1|     25.0|
5.4|  0|
| 17|          4|       1|        0.9|     25.6|
5.1|  0|
| 18|          5|       0|        1.2|     24.2|
11.1|  1|
| 19|          1|       2|        2.2|     25.4|
3.5|  0|
+---+-----------+--------+----------+---------+-------
-------+---+
only showing top 20 rows

root
 |-- _c0: integer (nullable = true)
 |-- SHOT_NUMBER: integer (nullable = true)
 |-- DRIBBLES: integer (nullable = true)
 |-- TOUCH_TIME: double (nullable = true)
 |-- SHOT_DIST: double (nullable = true)
 |-- CLOSE_DEF_DIST: double (nullable = true)
 |-- FGM: integer (nullable = true)
```

In [52]:

```
#Combine into Assembler and tranasform the dataset,and check the ou
assembler= VectorAssembler(inputCols=['SHOT_NUMBER','DRIBBLES', 'TO
output=assembler.transform(dataset)
output.show()
```

```
+---+-----------+--------+----------+---------+-------
-------+---+--------------------+
|_c0|SHOT_NUMBER|DRIBBLES|TOUCH_TIME|SHOT_DIST|CLOSE_D
```

```
EF_DIST|FGM|          Attributes|
+---+-----------+--------+----------+---------+-------
-------+---+--------------------+
|  0|          1|       2|       1.9|      7.7|
1.3|  1|[1.0,2.0,1.9,7.7,...|
|  1|          2|       0|       0.8|     28.2|
6.1|  0|[2.0,0.0,0.8,28.2...|
|  2|          3|       3|       2.7|     10.1|
0.9|  0|[3.0,3.0,2.7,10.1...|
|  3|          4|       2|       1.9|     17.2|
3.4|  0|[4.0,2.0,1.9,17.2...|
|  4|          5|       2|       2.7|      3.7|
1.1|  0|[5.0,2.0,2.7,3.7,...|
|  5|          6|       2|       4.4|     18.4|
2.6|  0|[6.0,2.0,4.4,18.4...|
|  6|          7|      11|       9.0|     20.7|
6.1|  0|[7.0,11.0,9.0,20....|
|  7|          8|       3|       2.5|      3.5|
2.1|  1|[8.0,3.0,2.5,3.5,...|
|  8|          9|       0|       0.8|     24.6|
7.3|  0|[9.0,0.0,0.8,24.6...|
|  9|          1|       0|       1.1|     22.4|
19.8|  0|[1.0,0.0,1.1,22.4...|
| 10|          2|       8|       7.5|     24.5|
4.7|  0|[2.0,8.0,7.5,24.5...|
| 11|          3|      14|      11.9|     14.6|
1.8|  1|[3.0,14.0,11.9,14...|
| 12|          4|       2|       2.9|      5.9|
5.4|  1|[4.0,2.0,2.9,5.9,...|
| 13|          1|       0|       0.8|     26.4|
4.4|  0|[1.0,0.0,0.8,26.4...|
| 14|          1|       0|       0.5|     22.8|
5.3|  0|[1.0,0.0,0.5,22.8...|
| 15|          2|       3|       2.7|     24.7|
5.6|  1|[2.0,3.0,2.7,24.7...|
| 16|          3|       6|       5.1|     25.0|
5.4|  0|[3.0,6.0,5.1,25.0...|
| 17|          4|       1|       0.9|     25.6|
5.1|  0|[4.0,1.0,0.9,25.6...|
| 18|          5|       0|       1.2|     24.2|
11.1|  1|[5.0,0.0,1.2,24.2...|
| 19|          1|       2|       2.2|     25.4|
3.5|  0|[1.0,2.0,2.2,25.4...|
+---+-----------+--------+----------+---------+-------
-------+---+--------------------+
only showing top 20 rows
```

In [53]:

```
#finalized data in two attributes
finalized_data=output.select('Attributes','FGM')
finalized_data.show()
finalized_data.count()
```

```
+--------------------+---+
|          Attributes|FGM|
+--------------------+---+
|[1.0,2.0,1.9,7.7,...|  1|
|[2.0,0.0,0.8,28.2...|  0|
|[3.0,3.0,2.7,10.1...|  0|
|[4.0,2.0,1.9,17.2...|  0|
|[5.0,2.0,2.7,3.7,...|  0|
|[6.0,2.0,4.4,18.4...|  0|
|[7.0,11.0,9.0,20....|  0|
|[8.0,3.0,2.5,3.5,...|  1|
|[9.0,0.0,0.8,24.6...|  0|
|[1.0,0.0,1.1,22.4...|  0|
|[2.0,8.0,7.5,24.5...|  0|
|[3.0,14.0,11.9,14...|  1|
|[4.0,2.0,2.9,5.9,...|  1|
|[1.0,0.0,0.8,26.4...|  0|
|[1.0,0.0,0.5,22.8...|  0|
|[2.0,3.0,2.7,24.7...|  1|
|[3.0,6.0,5.1,25.0...|  0|
|[4.0,1.0,0.9,25.6...|  0|
|[5.0,0.0,1.2,24.2...|  1|
|[1.0,2.0,2.2,25.4...|  0|
+--------------------+---+
only showing top 20 rows
```

Out[53]:

```
124711
```

In [0]:

```
#Train_test_split and Logisitic Regressor
train_data,test_data=finalized_data.randomSplit([0.7,0.3])
```

In [0]:

```
#Regressor to fit the train data
lg=LogisticRegression(featuresCol='Attributes',labelCol='FGM',maxIt
regressor=lg.fit(train_data)
```

In [56]:

```
#Prediction and result
pred=regressor.evaluate(test_data)
pred.predictions.show(20)
```

```
+--------------------+---+--------------------+-------
-------------+----------+
|          Attributes|FGM|       rawPrediction|
probability|prediction|
+--------------------+---+--------------------+-------
-------------+----------+
|[1.0,0.0,0.1,0.5,...|  1|[-12.545659115257...|[3.5603
1093516874...|       1.0|
|[1.0,0.0,0.1,1.2,...|  0|[3.63384294037717...|[0.9742
6528916004...|       0.0|
|[1.0,0.0,0.1,1.7,...|  0|[3.58653331856008...|[0.9730
5212790811...|       0.0|
|[1.0,0.0,0.1,1.7,...|  0|[3.76145755348573...|[0.9772
7844445508...|       0.0|
|[1.0,0.0,0.1,1.9,...|  0|[3.80296207682411...|[0.9781
8203515465...|       0.0|
|[1.0,0.0,0.1,2.0,...|  1|[-12.719392387058...|[2.9925
1801470323...|       1.0|
|[1.0,0.0,0.1,2.1,...|  0|[3.62183575571168...|[0.9739
6251928400...|       0.0|
|[1.0,0.0,0.1,2.2,...|  0|[3.76185454119381...|[0.9772
8725801080...|       0.0|
|[1.0,0.0,0.1,2.4,...|  0|[4.10550092485831...|[0.9837
8548313934...|       0.0|
|[1.0,0.0,0.1,2.5,...|  0|[3.72074700556351...|[0.9763
5667209826...|       0.0|
|[1.0,0.0,0.1,2.5,...|  1|[-12.416853539024...|[4.0497
4116496181...|       1.0|
|[1.0,0.0,0.1,2.7,...|  0|[3.82586034160213...|[0.9786
6541467288...|       0.0|
|[1.0,0.0,0.1,2.8,...|  1|[-12.44205887148,...|[3.9489
4216822593...|       1.0|
|[1.0,0.0,0.1,2.9,...|  1|[-12.349746695523...|[4.3308
3139471464...|       1.0|
|[1.0,0.0,0.1,3.0,...|  1|[-12.432358754491...|[3.9874
3360038396...|       1.0|
|[1.0,0.0,0.1,3.2,...|  1|[-12.406756434328...|[4.0908
3879258166...|       1.0|
|[1.0,0.0,0.1,3.3,...|  1|[-12.219031039321...|[4.9356
0166323378...|       1.0|
|[1.0,0.0,0.1,3.4,...|  1|[-12.126718863364...|[5.4129
0697224346...|       1.0|
|[1.0,0.0,0.1,4.6,...|  0|[3.91006035140315...|[0.9803
5439243337...|       0.0|
|[1.0,0.0,0.1,6.4,...|  0|[4.36311207732392...|[0.9874
2155073250...|       0.0|
+--------------------+---+--------------------+-------
-------------+----------+
only showing top 20 rows
```

In [0]:

```
coeff=regressor.coefficients
intercept=regressor.intercept
```

In [58]:

```
coeff
```

Out[58]:

```
DenseVector([-0.1502, 0.058, -0.2402, -0.128, -0.159,
16.2966])
```

In [59]:

```
intercept
```

Out[59]:

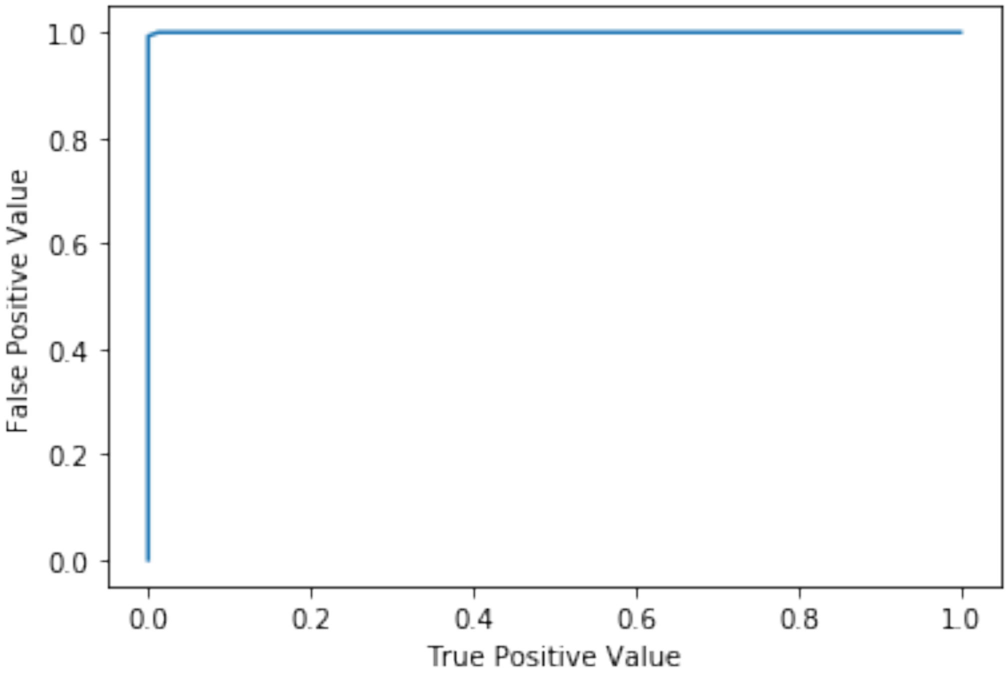
```
-2.9879468558313693
```

In [60]:

```
#Using model's summary to build a roc curve
summary=regressor.summary
ROC=summary.roc.toPandas()
##build ROC Curve
plt.plot(ROC['FPR'],ROC['TPR'])
#labels
plt.xlabel('True Positive Value')
plt.ylabel('False Positive Value')
#AUC value(on top)
summary.areaUnderROC
```

Out[60]:

```
0.9999554931353026
```



In [0]:

```
```