In [1]:
```
import numpy as np
```

```
Q1.
    a. Please download the data from http://archive.ics.uci.edu/ml/datasets/Liver+Disorders  (http://archive.ics.uci.edu/ml/datasets/Liver+Disorders)
    b. Get to know about the features
        i. given data set has 6 attributes and 1 output varaible
    c. Find me the two most corelated feature out of 6 attributes with the output varaible
```

Understanding the Data

In [2]:
```
import pandas as pd
df = pd.read_csv('bupa.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	mcv	alkphos	sgpt	sgot	gammagt	drinks	selector
0	85	92	45	27	31	0.0	1
1	85	64	59	32	23	0.0	2
2	86	54	33	16	54	0.0	2
3	91	78	34	24	36	0.0	2
4	87	70	12	28	10	0.0	2

features :

mcv : mean corpuscular volume

alkphos: alkaline phosphotase

sgpt : alamine aminotransferase

sgot : aspartate aminotransferase

gammagt : gamma-glutamyl transpeptidase

drinks : number of half-pint equivalents of alcoholic beverages drunk per day

selector : field used to split data into two sets

In [4]:
```
pearson_corelation_matrix = df.corr(method='pearson')
pearson_corelation_matrix
```

Out[4]:

	mcv	alkphos	sgpt	sgot	gammagt	drinks	selector
mcv	1.000000	0.044103	0.147695	0.187765	0.222314	0.312680	-0.091070
alkphos	0.044103	1.000000	0.076208	0.146057	0.133140	0.100796	-0.098050
sgpt	0.147695	0.076208	1.000000	0.739675	0.503435	0.206848	-0.035009
sgot	0.187765	0.146057	0.739675	1.000000	0.527626	0.279588	0.157356
gammagt	0.222314	0.133140	0.503435	0.527626	1.000000	0.341224	0.146393
drinks	0.312680	0.100796	0.206848	0.279588	0.341224	1.000000	-0.022049
selector	-0.091070	-0.098050	-0.035009	0.157356	0.146393	-0.022049	1.000000

In [5]:
```
spearman_corelation_matrix = df.corr(method='spearman')
spearman_corelation_matrix
```

Out[5]:

	mcv	alkphos	sgpt	sgot	gammagt	drinks	selector
mcv	1.000000	0.045252	0.101325	0.106042	0.216296	0.320261	-0.102466
alkphos	0.045252	1.000000	0.137222	0.188140	0.156109	0.024078	-0.122227
sgpt	0.101325	0.137222	1.000000	0.570193	0.570833	0.150735	-0.134678
sgot	0.106042	0.188140	0.570193	1.000000	0.465419	0.254818	0.144640
gammagt	0.216296	0.156109	0.570833	0.465419	1.000000	0.341523	0.219611
drinks	0.320261	0.024078	0.150735	0.254818	0.341523	1.000000	0.038725
selector	-0.102466	-0.122227	-0.134678	0.144640	0.219611	0.038725	1.000000

Observations :

By observing above corelation matrices (pearson and spearman), we can say that

> **sgot** and **gammagt** are the two most corelated features with the output variable

Prove the corelation with Hypothesis testing

In [6]:
```
df.head()
```
Out[6]:

	mcv	alkphos	sgpt	sgot	gammagt	drinks	selector
0	85	92	45	27	31	0.0	1
1	85	64	59	32	23	0.0	2
2	86	54	33	16	54	0.0	2
3	91	78	34	24	36	0.0	2
4	87	70	12	28	10	0.0	2

In [7]:
```
# let's take get sgot and gammagt colums into an numpy array
sgot_col = df['sgot'].values
gammagt_col = df['gammagt'].values
selector_col = df['selector'].values
# just to check the array
sgot_col[:10]
```
Out[7]:
```
array([27, 32, 16, 24, 28, 17, 17, 11, 20, 19])
```

In [8]:
```
sgot_col.shape
```
Out[8]:
```
(345,)
```

In [9]:
```
# numpy.corrcoef returns Pearson product-moment correlation coefficients
print(np.corrcoef(sgot_col, selector_col)[0][1])
print(np.corrcoef(gammagt_col, selector_col)[0][1])
```
```
0.157355800969
0.146392523648
```

From the pearsons correlation coefficients, ***sgot and gammagt*** are _**positively related** to the ***output variable***, but ***weakly***.

Hypothesis testing:

Prove that the features are correlated with output varaible using null hypothsis test i.hint: consider 100 random samples from the data set and find out the correlation, repeat it for 50 times

Null Hypothesis:

H_0 : sgot and output variable are negatively corelated

In [10]:
```
indices = np.arange(345)
```

In [11]:
```
# for finding the p-value
count = 0
temp = 0
# for sgot and output variable
for i in range(100):
    # take 100 samples from dataset
    np.random.shuffle(indices)
    sample_indices = indices[:100]
    sgot_sample = sgot_col[sample_indices]
    output_sample = selector_col[sample_indices]
    pcc = np.corrcoef(sgot_sample, output_sample)[0][1]
    # check if pcc is negatively correlated or not
    if pcc<=0:
        count = count + 1

p_value = count/50
p_value
```

Out[11]: 0.06

Observation:

With p_value 0.02 (<0.05), we can say that NUll Hypothesis is true with probability 0.02.

ie., We can strongly reject Null Hypothesis.

Conclusion:

> ***sgot and output*** variable are ***correlated.***

Null Hypothesis:

H_0 : gammagt and output variable are not at all corelated

In [12]:
```
# for finding the p-value
count = 0
temp = 0
# for gammagt and output variable
for i in range(50):
    # take 100 samples from dataset
    np.random.shuffle(indices)
    sample_indices = indices[:100]
    gammagt_sample = gammagt_col[sample_indices]
    output_sample = selector_col[sample_indices]
    pcc = np.corrcoef(gammagt_sample, output_sample)[0][1]
    # check if pcc is negatively correlated or not
    if pcc<=0:
        count = count + 1

p_value = count/50
p_value
```

Out[12]: 0.06

Observation:

With p_value 0.04 (<0.05), we can say that NUll Hypothesis is true with probability 0.02.

ie., We can strongly reject Null Hypothesis.

Conclusion:

> ***gammagt and output*** variable are ***correlated.***

Q2.

```
a. Simulate the coin tossing by writing a function wich gives the output "head" with
50% chance and "tail" with 50% chance
b. call the same function 250 times, find out the number tests which gave the output
"head"
c. based on the result conclude that the coin is baised or not
d. prove your conclusion with the help of null hypothisis test
```

In [13]:
```
# performs the coin toss and returns heads or tails..
def CoinToss():
    r = np.random.random()
    if r<=0.5:
        # heads
        return 1
    else:
        # tails
        return 0

# experiment coin toss 250 times and return no. of heads out of 250
def Experiment():
    heads = 0
    for i in range(250):
        if CoinToss():
            heads = heads + 1
    return heads


# if we repeat this experiment multiple times, we can say biased or not with ce
rtain probability.
# This is not Hypothesis testing, We will do it in next section.
heads = Experiment()
if 120 <= heads <= 130:
    print('Coin is UnBiased')
else:
    print('Coin is Biased')
```

```
Coin is Biased
```

Hypothesis testing for Coin Toss

H_0 : The coin is biased. (Null Hypothesis)

H_1 : The coin is Unbiased

- We will repeat the experiment (tossing a coin 250 times), 1000 times.
    - if we can get more heads (say >=140), most of the times out of 1000 times, then we will say that, " *OUR NULL HYPOTHESIS IS TRUE* "

        > *The coin is Biased*

    - Otherwise, the coin is Unbiased.

In [14]:
```
# for hypothesis testing
biased = 0
for i in range(1000):
    heads = Experiment()
    if heads >= 140:
        biased = biased + 1

# calculate the p-value. ie., the prob that our null hypothesis is true..
p_value = biased/1000
p_value
```

Out[14]:
```
0.038
```

Observations:

- The probability that the coin is Biased (**p-value**) is very very less (.031).

> In General, if ***p-value <= 0.05*** , then we can strongly reject the Null Hypothesis.

Conclusion:

- Our Null Hypothesis is False

> **The coin is Unbiased**

Q3.

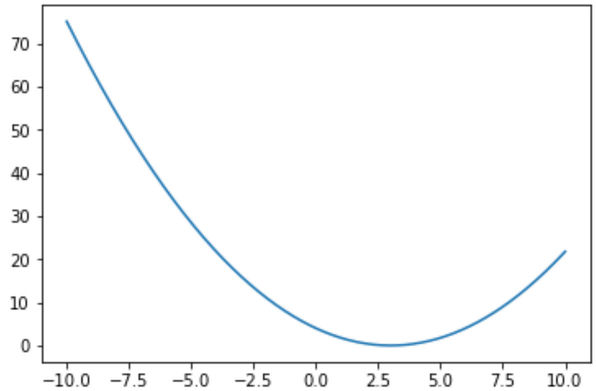
```
a. call the function genarate_data() to get two vectors, "X, Y = genarate_data()"
b. find out the trasofrmation of given vectors which will help us to find the correla
tion between X, Y with the help of techniques that are discussed in the class (Pearso
n Product Moment Correlation, Spearman rank Order Correlation)
    hint: use the techniques that are discussed in the class while solving "X^2 + Y^2
=a^2 (circle data)"
```

In [15]:
```
import numpy as np
import math
def genarate_data():
    X = np.linspace(-10,10, 500)
    Y = [(4 / 3 ** 2) * (x - 3) ** 2 for x in X]
    return X, Y
```

In [16]:
```
X, Y = genarate_data()
```

In [17]:
```
import matplotlib.pyplot as plt
```

In [18]:
```
plt.plot(X, Y)
plt.show()
```



Observations:

1. For this data, ***Pearson correlation coefficient*** might ***not*** give the best results, because the data is ***not linear***.

1. But we can easily rank the data, cause there is a _**unique value** for ***each x***. That ***eliminates*** the ***need of data transformation***, to apply ***Spearman corelation coefficient***.

Conclusion:

In [27]:
```
# Spearman rank-order correlation coefficient for this data is..
import scipy.stats as stats
result = stats.spearmanr(X,Y)
```

In [30]:
```
print('Spearman rank-order correlation coefficient of X and Y is {}. '.format(r
esult.correlation))
```

```
Spearman rank-order correlation coefficient of X and Y is -0.6555314221256885.
```