HILL AND VALLEY PREDICTION

Each record represents 100 points on a 2-D graph. When plotted in order, from 1-100,as the y-coordinate then the points will create either a hill(bump) or valley(dip)

1-100 :labeled"V##". Floating point values (numeric) - x valves

101 : labeled "class". Binary{0,1} representing valley , hill

Data is retrived from ybi data hub

Import Library

```
import pandas as pd
```

Import data

```
df = pd.read_csv('https://github.com/YBIFoundation/Dataset/raw/main/Hill%20Valley%20Dataset.csv')
```

```
df.describe()
```

	V1	V2	V3	V4	V5	V6	
count	1212.000000	1212.000000	1212.000000	1212.000000	1212.000000	1212.000000	1212.00
mean	8169.091881	8144.306262	8192.653738	8176.868738	8128.297211	8173.030008	8188.58
std	17974.950461	17881.049734	18087.938901	17991.903982	17846.757963	17927.114105	18029.56
min	0.920000	0.900000	0.850000	0.890000	0.880000	0.860000	0.87
25%	19.602500	19.595000	18.925000	19.277500	19.210000	19.582500	18.69
50%	301.425000	295.205000	297.260000	299.720000	295.115000	294.380000	295.93
75%	5358.795000	5417.847500	5393.367500	5388.482500	5321.987500	5328.040000	5443.97
max	117807.870000	108896.480000	119031.350000	110212.590000	113000.470000	116848.390000	115609.24

8 rows × 101 columns

```
df.columns
```

```
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       ...
       'V92', 'V93', 'V94', 'V95', 'V96', 'V97', 'V98', 'V99', 'V100',
       'Class'],
      dtype='object', length=101)
```

```
print(df.columns.tolist())
```

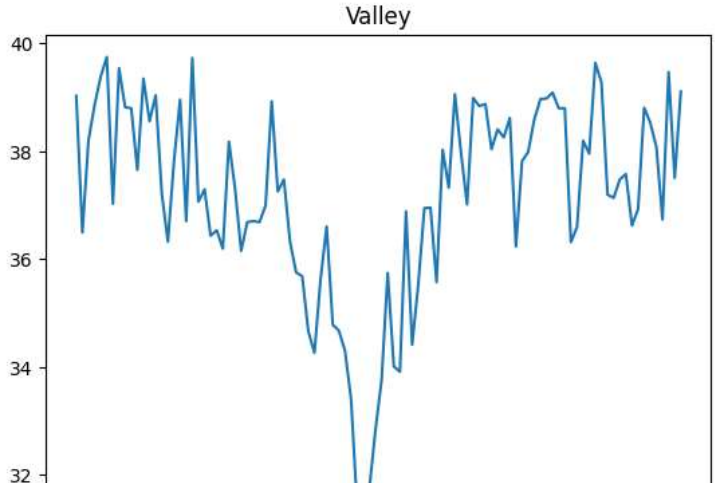
```
['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
```

```
y = df['Class']
x = df[['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20'
```

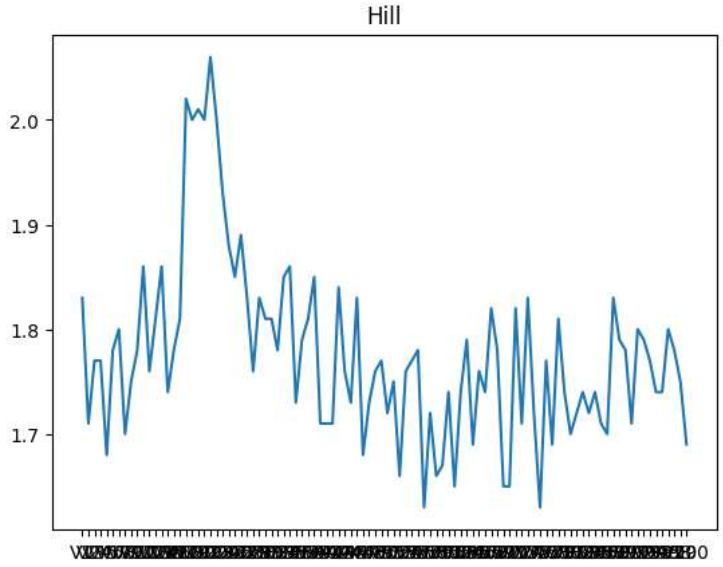
Data Visualization

```
import matplotlib.pyplot as plt
```

```
plt.plot(x.iloc[0,:])
plt.title('Valley');
```




```
plt.plot(x.iloc[1,:])
plt.title('Hill');
```



Data Preprocessing

Standardizaaation of datasets is a common requirement for many machine learning estimators implemented in scikit learn

they might behave badly if the individual features do not more or less look like standard normally distributed data : Gaussian with zero mean and unit variance

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

```
x = ss.fit_transform(x)
x
```

```
array([[-0.45248681, -0.45361784, -0.45100881, ..., -0.45609618,
        -0.45164274, -0.45545496],
       [-0.45455665, -0.45556372, -0.45302369, ..., -0.45821768,
        -0.45362255, -0.45755405],
       [ 3.33983504,  3.24466709,  3.58338069, ...,  3.5427869 ,
         3.27907378,  3.74616847],
       ...,
       [ 0.11084204,  0.0505953 ,  0.04437307, ...,  0.12533312,
         0.04456025,  0.06450317],
       [-0.45272112, -0.45369729, -0.45118691, ..., -0.45648861,
        -0.45190136, -0.45569511],
       [ 0.01782872, -0.02636986,  0.05196137, ...,  0.03036056,
         0.01087365,  0.03123129]])
```

```
x.shape
```

```
(1212, 100)
```

TRAIN TEST SPLIT

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.7,random_state=2529)

x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((848, 100), (364, 100), (848,), (364,))
```

Select model

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

Train/fit model

```
model.fit(x_train,y_train)
```

```
LogisticRegression()
```

Prediction

```
y_pred = model.predict(x_test)
y_pred
```

```
array([1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0])
```

Accuracy

```
from sklearn.metrics import confusion_matrix,classification_report

print(confusion_matrix(y_test, y_pred))
```

```
[[176   4]
 [ 92  92]]
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.66      0.98      0.79       180
               1       0.96      0.50      0.66       184

        accuracy                           0.74       364
       macro avg       0.81      0.74      0.72       364
    weighted avg       0.81      0.74      0.72       364
```

Therefore we were able to predict the hill and valley with accuracy , vizualization, and preprocessing