# Hill and Valley Prediction using Logistic Regression

**Get Understanding about Data set**

Each record represents 100 points on a two-dimensional graph. When plotted in order (from 1 through 100) as the Y coordinate, the points will create either a Hill (a "bymp" in the terrain) or a Valley (a "dip" in the terrain). See the original source for some examples of these graphs.

1-100: Labeled "V##", Floating point values (numeric), the X-values. 101: Labeled "Class", Binary {0,1} representing {valley, hill}.

# Import Library

```
import pandas as pd
```

```
import numpy as np
```

# Import CSV as DataFrame

```
df = pd.read_csv(r'/content/Hill Valley Dataset.csv')
```

Saved successfully!

# Get the First Five Rows of DataFrame

```
df.head()
```

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 39.02 | 36.49 | 38.20 | 38.85 | 39.38 | 39.74 | 37.02 | 39.53 | 38.81 |
| **1** | 1.83 | 1.71 | 1.77 | 1.77 | 1.68 | 1.78 | 1.80 | 1.70 | 1.75 |
| **2** | 68177.69 | 66138.42 | 72981.88 | 74304.33 | 67549.66 | 69367.34 | 69169.41 | 73268.61 | 74465.84 |

## Get Information of DataFrame

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1212 entries, 0 to 1211
Columns: 101 entries, V1 to Class
dtypes: float64(100), int64(1)
memory usage: 956.5 KB
```

## Get the Summary Statistics

```
df.describe()
```

| | V1 | V2 | V3 | V4 | V5 | |
|---|---|---|---|---|---|---|
| **count** | 1212.000000 | 1212.000000 | 1212.000000 | 1212.000000 | 1212.000000 | 1212.0( |
| **mean** | 8169.091881 | 8144.306262 | 8192.653738 | 8176.868738 | 8128.297211 | 8173.0 |
| **std** | 17974.950461 | 17881.049734 | 18087.938901 | 17991.903982 | 17846.757963 | 17927.1 |
| **min** | 0.920000 | 0.900000 | 0.850000 | 0.890000 | 0.880000 | 0.8 |
| [illegible] | [illegible] | [illegible]5000 | 18.925000 | 19.277500 | 19.210000 | 19.5 |
| **50%** | 301.425000 | 295.205000 | 297.260000 | 299.720000 | 295.115000 | 294.3 |
| **75%** | 5358.795000 | 5417.847500 | 5393.367500 | 5388.482500 | 5321.987500 | 5328.0 |
| **max** | 117807.870000 | 108896.480000 | 119031.350000 | 110212.590000 | 113000.470000 | 116848.3 |

Saved successfully!

8 rows × 101 columns

## Get Column Names

```
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
```

## Get Shape of DataFrame

```
df.shape
```

```
(1212, 101)
```

## Get Unique Values (Class or Label) in y Variable

```
df['Class'].value_counts()
```

```
0    606
1    606
Name: Class, dtype: int64
```

```
df.groupby('Class').mean()
```

Saved successfully!

| Class | V1 | V2 | V3 | V4 | V5 | V6 | |
|---|---|---|---|---|---|---|---|
| **0** | 7913.333251 | 7825.339967 | 7902.497294 | 7857.032079 | 7775.610198 | 7875.436337 | 7804. |
| **1** | 8424.850512 | 8463.272558 | 8482.810182 | 8496.705396 | 8480.984224 | 8470.623680 | 8572. |

2 rows × 100 columns

# Define y (dependent or label or target variable) and X (independent or features or attribute Variable)

```
y = df['Class']
```

```
y.shape
```

```
(1212,)
```

```
y
```

```
0       0
1       1
2       1
3       0
4       0
       ..
1207    1
1208    0
1209    1
1210    1
1211    0
Name: Class, Length: 1212, dtype: int64
```

```
X = df.drop('Class', axis=1)
```

```
X.shape
```

```
(1212, 100)
```

Saved successfully!

```
X
```

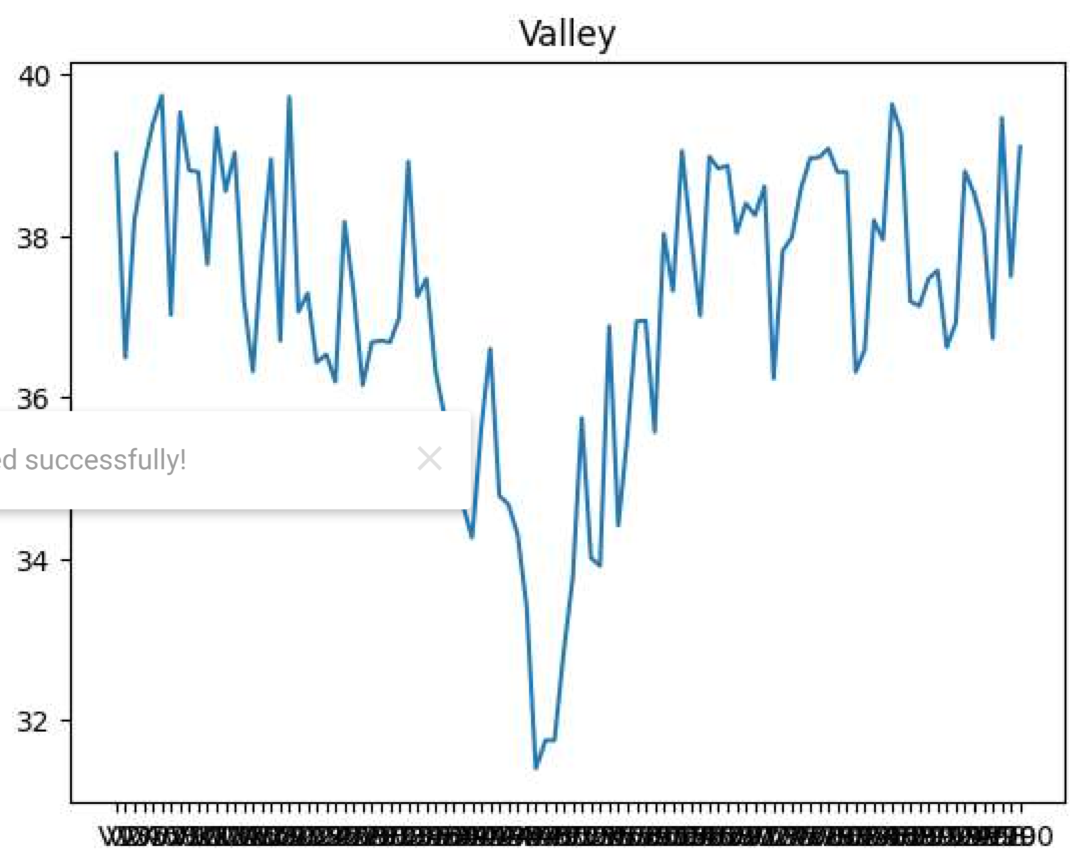
| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 39.02 | 36.49 | 38.20 | 38.85 | 39.38 | 39.74 | 37.02 | 39.53 | 38. |
| **1** | 1.83 | 1.71 | 1.77 | 1.77 | 1.68 | 1.78 | 1.80 | 1.70 | 1. |
| **2** | 68177.69 | 66138.42 | 72981.88 | 74304.33 | 67549.66 | 69367.34 | 69169.41 | 73268.61 | 74465. |
| **3** | 44889.06 | 39191.86 | 40728.46 | 38576.36 | 45876.06 | 47034.00 | 46611.43 | 37668.32 | 40980. |
| **4** | 5.70 | 5.40 | 5.28 | 5.38 | 5.27 | 5.61 | 6.00 | 5.38 | 5. |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |

## Get Plot of First Two Rows

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

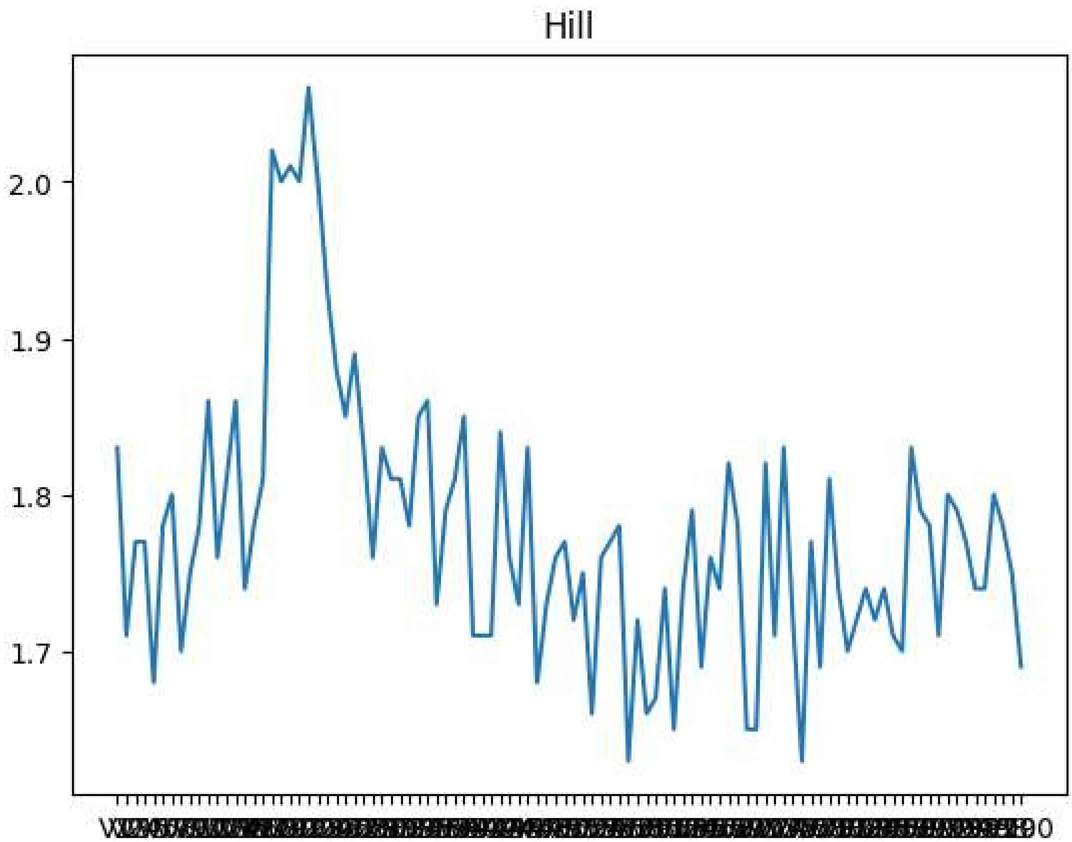
**1211** 8489.43 7672.98 9132.14 7985.73 8226.85 8554.28 8838.87 8967.24 8635

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




Saved successfully!

```
plt.plot(X.iloc[1,:])
plt.title("Hill");
```




## Get X Variables Standardized

Standardization of datasets is a common requirement for many machinelearning 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)
```

# Get Train Test Split

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, stratify = y, rand
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

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

# Get Model Train

```
from sklearn.linear_model import LogisticRegression
```

Saved successfully!

```
lr.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

# Get Model Prediction

```
y_pred = lr.predict(X_test)
```

```
y_pred.shape
```

```
(364,)
```

```
y_pred
```

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

## Get Probability of Each Predicted Class

```
lr.predict_proba(X_test)
```

Saved successfully!

```
       [0.50867123, 0.49132877],
       [0.63597662, 0.36402338],
       [0.50894067, 0.49105933],
       [0.53221095, 0.46778905],
       [0.4735918 , 0.5264082 ],
       [0.51131842, 0.48868158],
       [0.47711039, 0.52288961],
       [0.50836868, 0.49163132],
       [0.51791139, 0.48208861],
       [0.50929629, 0.49070371],
       [0.77824423, 0.22175577],
       [0.50878251, 0.49121749],
       [0.53444807, 0.46555193],
       [0.68401993, 0.31598007],
       [0.50837605, 0.49162395],
       [0.49955891, 0.50044109],
       [0.66733466, 0.33266534],
       [0.51287013, 0.48712987],
       [0.5100684 , 0.4899316 ],
       [0.49493863, 0.50506137],
       [0.50415413, 0.49584587],
       [0.50875424, 0.49124576],
       [0.50859968, 0.49140032],
       [0.51714632, 0.48285368],
       [0.49828539, 0.50171461],
       [0.50868242, 0.49131758],
       [0.5887849 , 0.4112151 ],
       [0.50578145, 0.49421855],
       [0.50864404, 0.49135596],
       [0.78691964, 0.21308036],
       [0.50865314, 0.49134686],
       [0.5085737 , 0.4914263 ],
       [0.48168718, 0.51831282],
       [0.50838591, 0.49161409],
       [0.05333433, 0.94666567]])
```

## Get Model Evaluation



```
from sklearn.metrics import confusion_matrix, classification_report
```

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

```
[[181   1]
 [106  76]]
```

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

```
              precision    recall  f1-score   support

           0       0.63      0.99      0.77       182
           1       0.99      0.42      0.59       182
```

```
    accuracy                          0.71       364
   macro avg       0.81      0.71      0.68       364
weighted avg       0.81      0.71      0.68       364
```

# Get Future Predictions

**Lets select a random sample from existing dataset as new value**

Steps to follow

1. Extract a random row using sample function
2. Seperate X and y
3. Standardize X
4. Predict

```
X_new = df.sample(1)
```

```
X_new
```

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | ... | V92 | V93 | V94 | V95 | V96 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **88** | 1.25 | 1.19 | 1.25 | 1.19 | 1.2 | 1.18 | 1.18 | 1.18 | 1.14 | 1.14 | ... | 1.19 | 1.17 | 1.08 | 1.04 | 1.05 |

1 rows × 101 columns

Saved successfully!

```
X_new.shape
```

```
(1, 101)
```

```
X_new = X_new.drop("Class", axis = 1)
```

```
X_new
```

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | ... | V91 | V92 | V93 | V94 | V95 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **88** | 1.25 | 1.19 | 1.25 | 1.19 | 1.2 | 1.18 | 1.18 | 1.18 | 1.14 | 1.14 | ... | 1.19 | 1.19 | 1.17 | 1.08 | 1.04 |

1 rows × 100 columns

```
X_new.shape
```

```
(1, 100)
```

```
X_new = ss.fit_transform(X_new)
```

```
y_pred_new = lr.predict(X_new)
```

```
y_pred_new
```

```
array([1])
```

```
lr.predict_proba(X_new)
```

```
array([[0.49714993, 0.50285007]])
```

Saved successfully!

0s completed at 8:26 PM