| |
|---|
| Experiment No. 6 |
| Apply Boosting Algorithm on Adult Census Income Dataset and analyze the performance of the model |
| Date of Performance: |
| Date of Submission: |

**Aim:** Apply Boosting algorithm on Adult Census Income Dataset and analyze the performance of the model.

**Objective:** Apply Boosting algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

**Theory:**

Suppose that as a patient, you have certain symptoms. Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnosis they have made. The final diagnosis is then a combination of the weighted diagnosis. This is the essence behind boosting.

Algorithm: Adaboost- A boosting algorithm—create an ensemble of classifiers. Each one

gives a weighted vote.

**Input:**

- D , a set of d class labelled training tuples
- k, the number of rounds (one classifier is generated per round)
- a classification learning scheme

**Output:** A composite model

**Method**

1. Initialize the weight of each tuple in D is 1/d
2. For i=1 to k do // for each round
3. Sample D with replacement according to the tuple weights to obtain $D_i$
4. Use training set $D_i$ to derive a model $M_i$
5. Computer error($M_i$), the error rate of $M_i$
6. Error($M_i$)=$\sum w_j$*err($X_j$)
7. If Error($M_i$)>0.5 then
8. Go back to step 3 and try again
9. endif
10. for each tuple in $D_i$ that was correctly classified do
11. Multiply the weight of the tuple by error(Mi)/(1-error($M_i$)
12. Normalize the weight of each tuple
13. end for

**To use the ensemble to classify tuple X**

1. Initialize the weight of each class to 0
2. for i=1 to k do // for each classifier
3. $w_i$=log((1-error($M_i$))/error($M_i$))//weight of the classifiers vote
4. C=$M_i$(X) // get class prediction for X from $M_i$
5. Add $w_i$ to weight for class C
6. end for
7. Return the class with the largest weight.

**Dataset:**

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.
Attribute Information:
Listing of attributes:

>50K, <=50K.

age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.
capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad &Tobago, Peru, Hong, Holand-Netherlands.

**Conclusion:**

1.In the conducted experiment, the XGBoost boosting algorithm yielded the following results: an accuracy of 87%, precision of 88%, recall of 94%, and an f1-score of 91%.

2.The application of the XGBoost algorithm to the Adult Income dataset has demonstrated its reliability and capacity to handle intricate data. This analysis clearly showcases XGBoost' s effectiveness in boosting weaker models and optimizing predictive accuracy, particularly in the context of income level predictions, which holds significance in various socio-economic applications.

3.When comparing the outcomes of applying boosting and random forest algorithms to the Adult Census Income Dataset, it's crucial to consider the trade-offs. Boosting typically offers superior predictive accuracy, especially for complex datasets, albeit potentially sacrificing some interpretability. Conversely, random forests maintain competitive accuracy while retaining better interpretability and a higher resistance to overfitting.

```python
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import
train_test_split,cross_val_score,KFold,GridSearchCV
from sklearn.metrics import
confusion_matrix,classification_report,accuracy_score

dataset=pd.read_csv("/content/drive/MyDrive/dataset/adult.csv")

print(dataset.isnull().sum())
print(dataset.dtypes)
```

```
age                 0
workclass           0
fnlwgt              0
education           0
educational-num     0
marital-status      0
occupation          0
relationship        0
race                0
gender              0
capital-gain        0
capital-loss        0
hours-per-week      0
native-country      0
income              0
dtype: int64
age                  int64
workclass           object
fnlwgt               int64
education           object
educational-num      int64
marital-status      object
occupation          object
relationship        object
race                object
gender              object
capital-gain         int64
capital-loss         int64
hours-per-week       int64
native-country      object
```

```
income             object
dtype: object
```

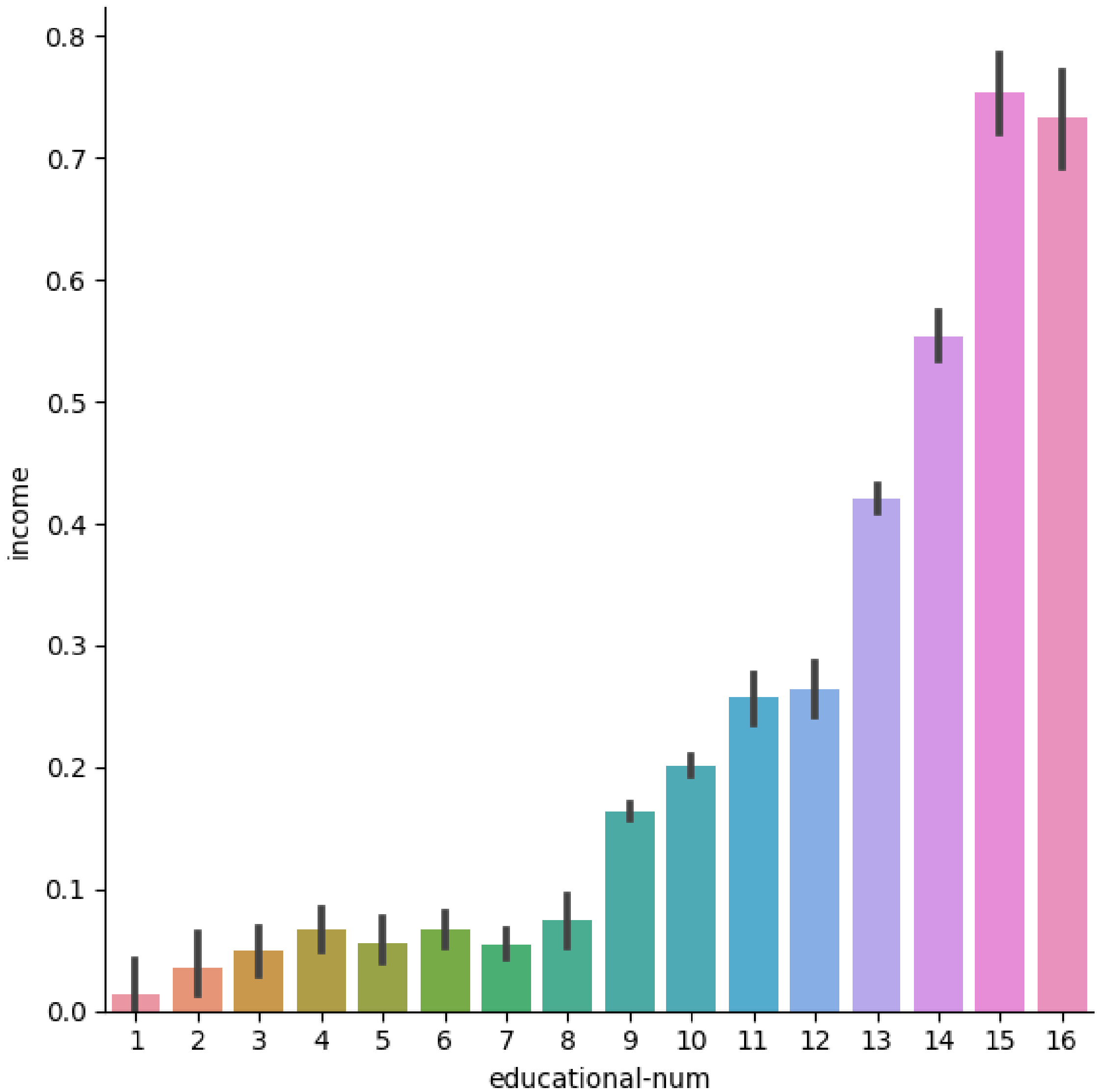
```
dataset.head()
```

```
   age  workclass  fnlwgt     education  educational-num      marital-
status  \
0   25    Private  226802          11th                7       Never-
married
1   38    Private   89814       HS-grad                9  Married-civ-
spouse
2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-
spouse
3   44    Private  160323  Some-college               10  Married-civ-
spouse
4   18          ?  103497  Some-college               10       Never-
married

          occupation relationship   race  gender  capital-gain
capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0
0
1    Farming-fishing      Husband  White    Male             0
0
2    Protective-serv      Husband  White    Male             0
0
3  Machine-op-inspct      Husband  Black    Male          7688
0
4                  ?    Own-child  White  Female             0
0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
```

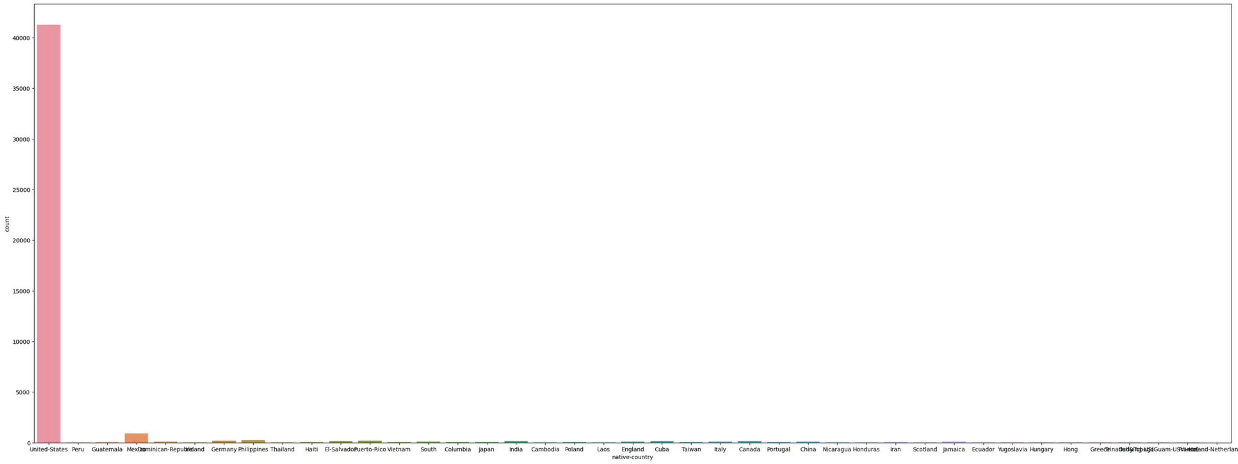
```
#removing '?' containing rows
dataset = dataset[(dataset != '?').all(axis=1)]
#label the income objects as 0 and 1
dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1})
```

```
<ipython-input-18-39ed73805135>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
```

```
returning-a-view-versus-a-copy
  dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1})

sns.catplot(x='educational-
num',y='income',data=dataset,kind='bar',height=6)
plt.show()
```




```
#explore which country do most people belong
plt.figure(figsize=(38,14))
sns.countplot(x='native-country',data=dataset)
plt.show()
```

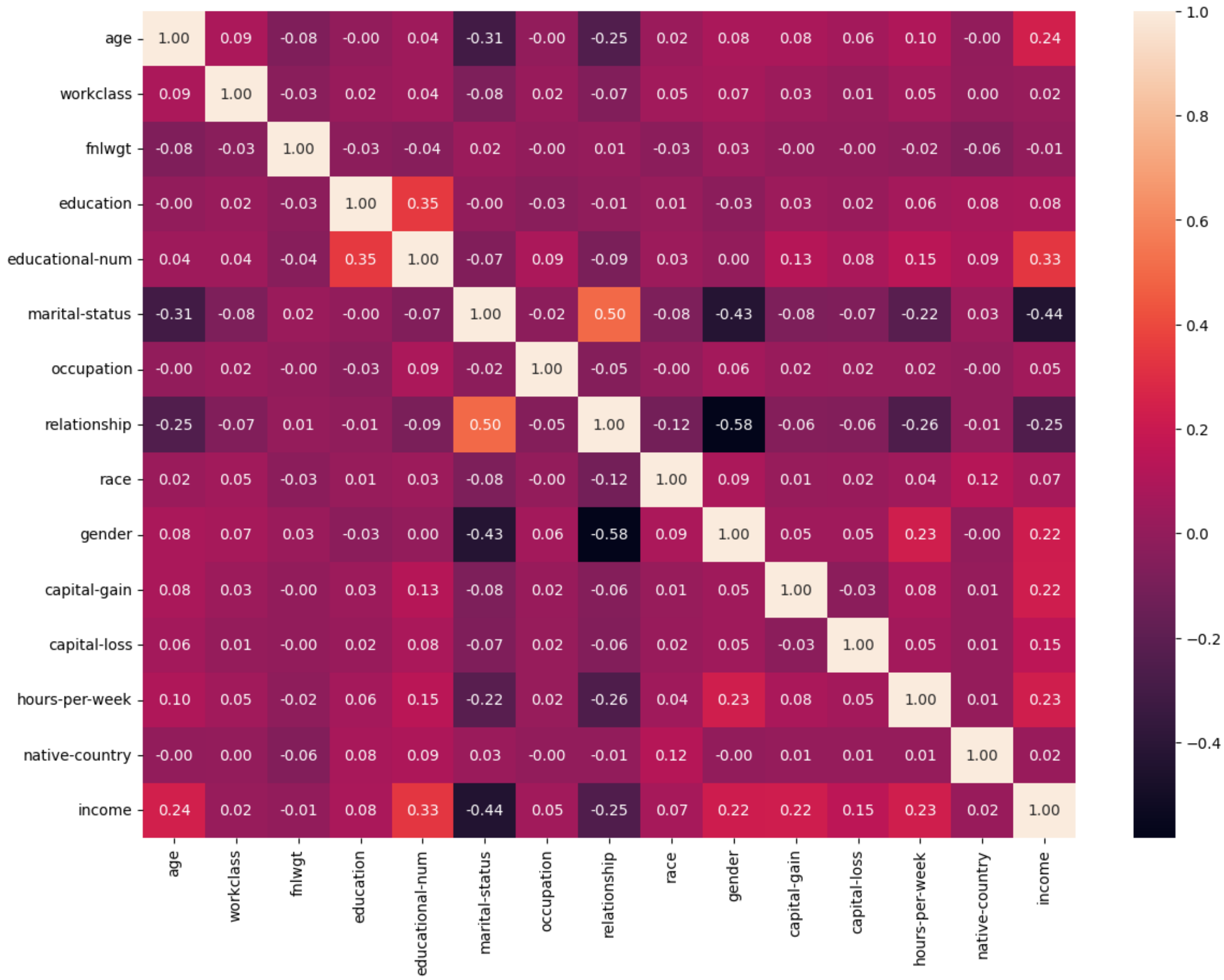



```
dataset['marital-status']=dataset['marital-status'].map({'Married-civ-
spouse':'Married', 'Divorced':'Single', 'Never-married':'Single',
'Separated':'Single',
'Widowed':'Single', 'Married-spouse-absent':'Married', 'Married-AF-
spouse':'Married'})

for column in dataset:
    enc=LabelEncoder()
    if dataset.dtypes[column]==np.object:
        dataset[column]=enc.fit_transform(dataset[column])
```

```
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
```

```
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
<ipython-input-24-5d7d7fe4d7c0>:3: DeprecationWarning: `np.object` is
a deprecated alias for the builtin `object`. To silence this warning,
use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  if dataset.dtypes[column]==np.object:
```

```python
plt.figure(figsize=(14,10))
sns.heatmap(dataset.corr(),annot=True,fmt='.2f')
plt.show()
```




```
dataset=dataset.drop(['relationship','education'],axis=1)

dataset=dataset.drop(['occupation','fnlwgt','native-country'],axis=1)

print(dataset.head())
```

```
   age  workclass  educational-num  marital-status  race  gender  \
0   25          2                7               1     2       1
1   38          2                9               0     4       1
2   28          1               12               0     4       1
3   44          2               10               0     2       1
5   34          2                6               1     4       1

   capital-gain  capital-loss  hours-per-week  income
0             0             0              40       0
1             0             0              50       0
2             0             0              40       1
3          7688             0              40       1
5             0             0              30       0
```

```
X=dataset.iloc[:,0:-1]
y=dataset.iloc[:,-1]
print(X.head())
print(y.head())
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,shuf
fle=False)
```

```
   age  workclass  educational-num  marital-status  race  gender  \
0   25          2                7               1     2       1
1   38          2                9               0     4       1
2   28          1               12               0     4       1
3   44          2               10               0     2       1
5   34          2                6               1     4       1

   capital-gain  capital-loss  hours-per-week
0             0             0              40
1             0             0              50
2             0             0              40
3          7688             0              40
5             0             0              30
0    0
1    0
2    1
3    1
5    0
Name: income, dtype: int64
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
from xgboost import XGBClassifier
classifier=XGBClassifier()
classifier.fit(X_train,y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan,
monotone_constraints=None,
```

```
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
from sklearn.metrics import confusion_matrix,accuracy_score ,
classification_report
y_pred=classifier.predict(X_test)
cm=confusion_matrix(y_test,y_pred)
print(cm)
accuracy_score(y_test,y_pred)
```

```
[[10532   638]
 [ 1375  2379]]
```

```
0.8651165907263468
```

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

```
              precision    recall  f1-score   support

           0       0.88      0.94      0.91     11170
           1       0.79      0.63      0.70      3754

    accuracy                           0.87     14924
   macro avg       0.84      0.79      0.81     14924
weighted avg       0.86      0.87      0.86     14924
```

```
from sklearn.model_selection import cross_val_score
accuracies=cross_val_score(estimator=classifier,X=X_train,y=y_train,cv
=10)
print('Accuracy: {:.2f} Standard Deviation:
{:.2f}'.format(accuracies.mean()*100,accuracies.std()*100))
```

```
Accuracy: 85.85 Standard Deviation: 0.36
```