Features Selection Based on Uni-Variante ROC_AUC for Classification and MSE for Regression.

What is the ROC_AUC ?

- AUC :- Area Under the Curve.
- AUROC :- Area Under the Receiver Operating Characteristic curve.
- MSE :- Mean square error
- ROC is a probability curve and AUC represents degree or measure of separability.
- Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between patients with disease and no disease.
- An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever.

- The receiver operating characteristics(ROC) curve is well known in evaluating classification performance.Owing to its superiority in dealing with imbalaced and cost-sensitive data,the ROC curve has been exploited as a popular metric to evaluate the ML models.

- ROC_CURVE is drawn against the `sensitivity and 1-specificity` area under ROC curve shows the AUC which is a metric.
- Exsisting ROC based on feature selection apporoches are simple and effecive in individual feature selection.This is uni_variate feature selection technique.That means while selecting the one feature it doesnt found out its effect any on the current features.That means we find the features important only based on AUC and ROC curve charectestics or metrics on UC metric.

- This is been widely used to determine the classification accuracy in supervised learning.

- This kind of the ROC_AUC classification can be used for the binary although it used for many classification (i.e.more than the binary) but this will be the difficult to use.

Use of ROC_AUC in Classification Problem.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

In [8]:
```
from sklearn.feature_selection import VarianceThreshold
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,roc_auc_score
```

In [7]:
```
data = pd.read_csv('titanic.csv')
data.head()
```

Out[7]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [9]:
```
data.dropna(inplace=True)
```

In [10]:
```
label = LabelEncoder()
data['sex'] = label.fit_transform(data['sex'])
data['embarked'] = label.fit_transform(data['embarked'])
data['class'] = label.fit_transform(data['embarked'])
data['who']=label.fit_transform(data['who'])
data['adult_male']=label.fit_transform(data['adult_male'])
data['embark_town']=label.fit_transform(data['embark_town'])
data['alive']=label.fit_transform(data['alive'])
data['alone'] = label.fit_transform(data['alone'])
```

In [15]:
```
del data['deck']
```

In [16]:
```
x = data.drop(labels='alive',axis=1)
y= data['alive']

x.shape,y.shape
```

Out[16]:
```
((182, 13), (182,))
```

In [17]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size = 0.8,random_state =0)

x_train.shape,x_test.shape
```

Out[17]:
```
((145, 13), (37, 13))
```

In [ ]:
```
#remove the constant and quasi constant
```

In [18]:
```
constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(x_train)
constant_filter.fit(x_test)
x_train_fiter = constant_filter.transform(x_train)
x_test_filter = constant_filter.transform(x_test)

x_train_fiter.shape,x_test_filter.shape
```

Out[18]:
```
((145, 13), (37, 13))
```

In [ ]:
```
#we want to remove duplicate data
```

In [19]:
```
x_train_T = x_train_fiter.T
x_test_T  = x_test_filter.T

x_train_T = pd.DataFrame(x_train_T)
x_test_T  = pd.DataFrame(x_test_T)
```

In [ ]:

In [20]:
```
duplicate_feature = x_train_T.duplicated()
keep_them = [not index for index in duplicate_feature]

x_train_unique = x_train_T[keep_them].T
x_test_unique  = x_test_T[keep_them].T

x_train_unique.shape,x_test_unique.shape
```

Out[20]:
```
((145, 11), (37, 11))
```

Now Calculate ROC_AUC Score

In [21]:
```
#we need to create first upon the empty list like auc_roc
roc_auc = []
#It will read the one by one all the features from the columns of the X_train_unique.columns
for feature in x_train_unique.columns:
    clf = RandomForestClassifier(n_estimators=100,random_state=0)
#select the feature list from the x_train_unique and make it in the form of one diamensional array
    clf.fit(x_train_unique[feature].to_frame(),y_train)
    y_pred = clf.predict(x_test_unique[feature].to_frame())
#so,now we are appending value of roc_auc in the empty list
    roc_auc.append(roc_auc_score(y_test,y_pred))
```

In [22]:
```
print(roc_auc)
```

```
[1.0, 0.5, 0.8888888888888888, 0.5518518518518519, 0.5, 0.5, 0.5259259259259259, 0.48148148148148145, 0.9074074
074074074, 0.9074074074074074, 0.5]
```

In [24]:
```
#to put this array of roc_auc in the pandas Dataframe Series
roc_values = pd.Series(roc_auc)
roc_values.index = x_train_unique.columns
roc_values.sort_values(ascending= False,inplace = True)
```

- If we check then we find the series which is in sorted manner and also we will get the few values which are more than the 0.5
- and these values are the most important than the other becasue if we use the value which having magnitude equal to or lesser than the 0.5 that time
- It will give us random prediction and not accurate so to get the better result/prediction we have to choose more than the o.5 so that the predict should not go wrong or random.
- We are making here binary roc_auc technique so that those who are giving the 0.5 probablity will not be counted because they will not previde the imformation which is required to accurate prediction.

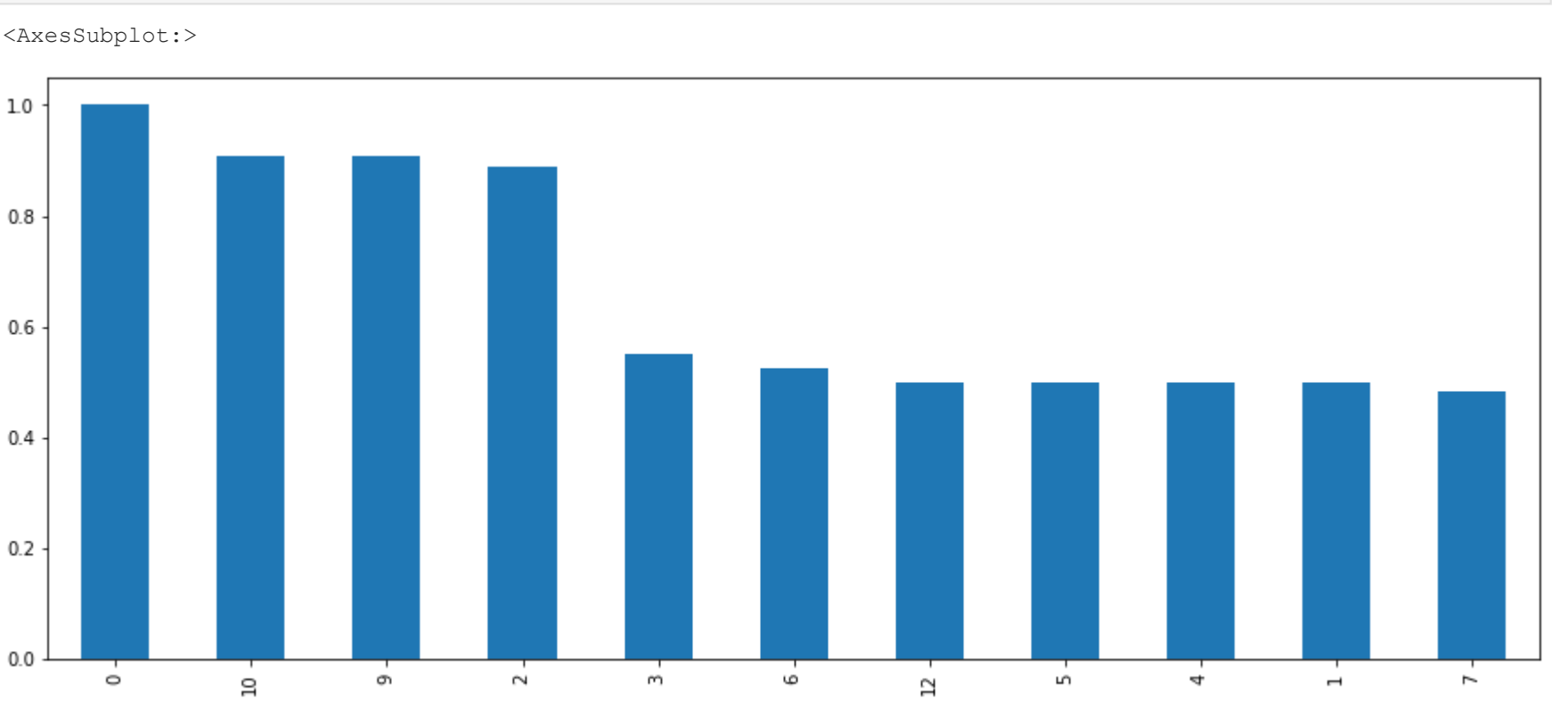
In [25]:
```
roc_values
```

Out[25]:
```
0     1.000000
10    0.907407
9     0.907407
2     0.888889
3     0.551852
6     0.525926
12    0.500000
5     0.500000
4     0.500000
1     0.500000
7     0.481481
dtype: float64
```

Those are the values are equal to the 0.5 and less than 0.5 are not required to classifiction.So for the we need to remove that kind of values.

In [26]:
```
roc_values.plot.bar(figsize = (15,6))
```

Out[26]:
```
<AxesSubplot:>
```



In [27]:
```
sel= roc_values[roc_values>0.5]
sel
```

Out[27]:
```
0     1.000000
10    0.907407
9     0.907407
2     0.888889
3     0.551852
6     0.525926
dtype: float64
```

In [28]:
```
len(sel)
```

Out[28]:
```
6
```

In [29]:
```
x_train_roc = x_train_unique[sel.index]
x_test_roc  = x_test_unique[sel.index]
```

Build the Model and compare the performance

In [31]:
```
def run_randomforest(x_train,x_test,y_train,y_test):
    clf = RandomForestClassifier(random_state=0,n_jobs=-1,n_estimators=1000)
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    print('Accuracy : ',accuracy_score(y_test,y_pred))
```

In [32]:
```
%%time
run_randomforest(x_train_roc,x_test_roc,y_train,y_test)
```

```
Accuracy :  1.0
Wall time: 2.01 s
```

In [34]:
```
x_train_roc.shape
```

Out[34]:
```
(145, 6)
```

- If we compare with the previous x_train,y_train then see.
- From that we can only say that there is no kind of impact on the accuracy but the time required to compute the things which is taken more than the x_train_roc,x_test_roc.

In [35]:
```
%%time
run_randomforest(x_train,x_test,y_train,y_test)
```

```
Accuracy :  1.0
Wall time: 1.9 s
```

- In the filtering method or any feature selection does not give us garantee of accuracy but it can give us time to compute the tthings which we want to calculate like accuracy.

- Diffrenet algorithm can give us diffrent output.If we here that first randomfoorest with the data x_train,x_test with the roc and then x_train,x_test with the no any substitution.The first one does not gievn the accuracy as much given by the roc data but it given the less computational time voice versa.
- According our requirement we choose the either first one or last one that having better accuracy but less high computational time.

Feature Selection using RMSE in Regression.

RMSE = Root mean square error

In [36]:
```
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

boston = datasets.load_boston()
```

In [37]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the targe
t.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity',
Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Internat
ional Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [38]:
```
x = pd.DataFrame(data=boston.data,columns=boston.feature_names)
x.head()
```

Out[38]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [39]:
```
y = boston.target
```

In [40]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size= 0.8,random_state =0)

x_train.shape,x_test.shape
```

Out[40]:
```
((404, 13), (102, 13))
```

In [41]:
```
#its method we have follow for univariant feature selection.
mse =[]
for feature in x_train.columns:
    clf = LinearRegression()
    clf.fit(x_train[feature].to_frame(),y_train)
    y_pred = clf.predict(x_test[feature].to_frame())
    mse.append(mean_squared_error(y_test,y_pred))
```

In [42]:
```
print(mse)
```

```
[76.38674157646072, 84.66034377707906, 77.02905244667242, 79.36120219345942, 76.95375968209432, 46.907351627395
3, 80.3915476111525, 82.61874125667717, 82.46499985731937, 78.30831374720843, 81.79497121208001, 77.75285601192
718, 46.33630536002592]
```

In [43]:
```
mse = pd.Series(mse)
mse.index = x_train.columns
mse.sort_values(ascending = False,inplace = True)
mse
```

Out[43]:
```
ZN         84.660344
DIS        82.618741
RAD        82.465000
PTRATIO    81.794971
AGE        80.391548
CHAS       79.361202
TAX        78.308314
B          77.752856
INDUS      77.029052
NOX        76.953760
CRIM       76.386742
RM         46.907352
LSTAT      46.336305
dtype: float64
```

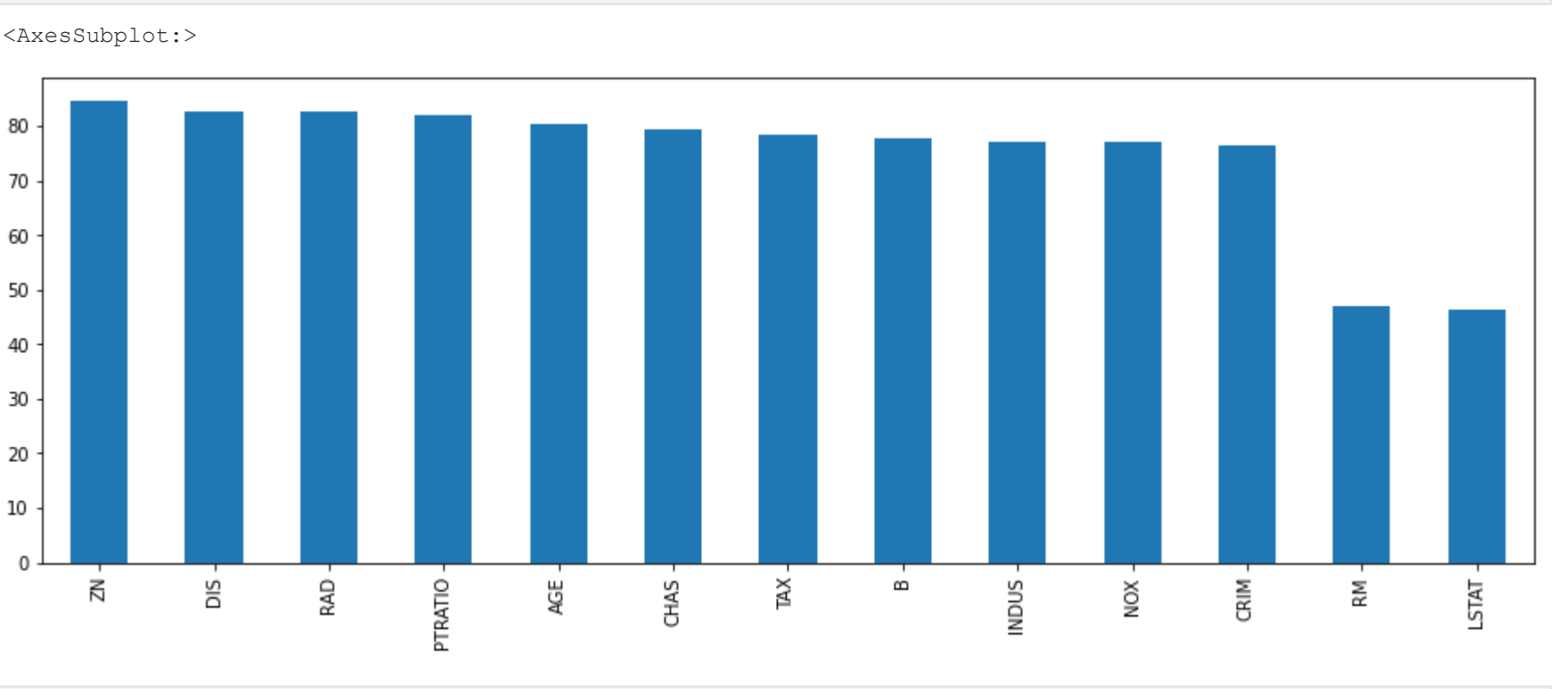
In [44]:
```
len(mse)
```

Out[44]:
```
13
```

Higher the mse means more the error and lower the mse means the low error if we can that the some features having the low mse those features are very important than the others.so we taking the two features to get the accuracy of linear regression.

In [45]:
```
mse.plot.bar(figsize = (15,5))
```

Out[45]:
```
<AxesSubplot:>
```



In [46]:
```
x_train_2 = x_train[['RM','LSTAT']]
x_test_2  = x_test[['RM','LSTAT']]
```

In [47]:
```
%%time
model = LinearRegression()
model.fit(x_train_2,y_train)
y_pred = model.predict(x_test_2)
print('r2 score :',r2_score(y_test,y_pred))
print('rmse :',np.sqrt(mean_squared_error(y_test,y_pred)))
print('sd of house price :',np.std(y))
```

```
r2 score : 0.5409084827186417
rmse : 6.114172522817781
sd of house price : 9.188011545278203
Wall time: 96.9 ms
```

- From the above we can understand that the root_mean_square_error is lesser than the standard deviation by selecting the two features.
- So,we can say that the present model is not bad model.

- If we do the prediction by using the original dataset then lets see what happen.

In [49]:
```
%%time
model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print('r2 score :',r2_score(y_test,y_pred))
print('rmse :',np.sqrt(mean_squared_error(y_test,y_pred)))
print('sd of house price :',np.std(y))
```

```
r2 score : 0.5892223849182507
rmse : 5.783509315085136
sd of house price : 9.188011545278203
Wall time: 98.2 ms
```

- Morever we can say that the feature selection by using the model it will give us of gurantee that we will get the accuracy or less error but will reduce our computational time.
- From the above we had tried first with the feature those who having the lower mean squared error it was the bad model afterwards
- we had taken the prediction on the original data in that we get the r2 score maximum than the previous but rmsq we lower than the previous one but computational time was quite high.

- So,we just say that we doesnt have gurantee to get the accuracy but it can be our suitability what we will prefered either time or accuracy.

- Sometimes,It depends on the model which we are using and data type and requirement.