IBM Innovation Centre for Education

Feature Selection : Correlation

Estimated time needed: **20** minutes

Objectives

After completing this lab you will have a good understanding in:

- Imporatance of Correlation
- Variance Value
- Why Constant Values are not important

This Notebook is created for Python Module

Feature Selection- With Correlation

The linear relationship between two or more variables is measured via correlation. We can forecast one variable based on another through correlation. Because the desirable variables have a strong correlation with the target, correlation can be used to select features. Furthermore, variables should be correlated with the target but should be uncorrelated among themselves.

We can anticipate one variable from another if the two are correlated. As a result, if two features are correlated, the model only actually requires one of them as the other does not provide any new information.Here, we'll make advantage of the Pearson Correlation.

In [42]:

```
#importing libraries
from sklearn.datasets import load_boston
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
#Loading the dataset
x = load_boston()
df = pd.DataFrame(x.data, columns = x.feature_names)
df["MEDV"] = x.target
```

```
/Users/sumitkumarshukla/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/deprecation.py:87: FutureWar
ning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Understand the Feature

In [3]:

```
print(data.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 tar
get.

    :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 Collinearit
y', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Inter
national Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [6]:

```
data.feature_names
```

Out[6]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [43]:

```
X = df.drop("MEDV",axis=1)   #Feature Matrix
y = df["MEDV"]
```

In [7]:

```
df.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
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	24.0
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	21.6
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	34.7
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	33.4
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	36.2

In [44]:

```
# separate dataset into train and test
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,random_state=0)

X_train.shape, X_test.shape, X_train.shape[0]+X_test.shape[0]
```

Out[44]:

```
((354, 13), (152, 13), 506)
```

Pearson Correlation Coefficient

The Pearson correlation coefficient (r) is the most widely used correlation coefficient and is known by many names:

```
Pearson’s r
Bivariate correlation
Pearson product-moment correlation coefficient (PPMCC)
The correlation coefficient
```

The Pearson correlation coefficient is a descriptive statistic, meaning that it summarizes the characteristics of a dataset. Specifically, it describes the **strength and direction of the linear relationship between two quantitative variables**. Although interpretations of the relationship strength (also known as effect size) vary between disciplines, the table below gives general rules of thumb:

Coeff Value	Strength	Slope Type
0.5 to 0.9	Strong +ve	upward
0.3 to 0.5	Moderate +ve	upward
0.0 to 0.3	Weak +ve	upward
C is 0.0	No Relation	
-0.0 to -0.3	Weak -ve	downhill
-0.3 to -0.5	Moderate -ve	downhill
-0.5 to -0.9	Strong -ve	downward

[Note]: The __Pearson correlation coefficient__ (r) is one of several correlation coefficients that you need to choose between when you want to measure a correlation. Range is between `-1 to 1.`

Formula

$$r=\frac{\sum\left(x_i-\bar{x}\right)\left(y_i-\bar{y}\right)}{\sqrt{\sum\left(x_i-\bar{x}\right)^2 \sum\left(y_i-\bar{y}\right)^2}}$$

- r = correlation coefficient
- x_{i} =   values of the x-variable in a sample
- $\bar{x}$ =   mean of the values of the x-variable
- y_{i} =   values of the y-variable in a sample
- $\bar{y}$ =   mean of the values of the y-variable

When and where to use Pearson Correlation

The **Pearson correlation coefficient** (r) is one of several correlation coefficients that you need to choose between when you want to measure a correlation. The Pearson correlation coefficient is a good choice when all of the following are true:

- Both variables are **quantitative:** You will need to use a different method if either of the variables is qualitative.
- The variables are **normally distributed:** You can create a histogram of each variable to verify whether the distributions are approximately normal. It’s not a problem if the variables are a little non-normal.
- The data have no **outliers:** Outliers are observations that don’t follow the same patterns as the rest of the data. A scatterplot is one way to check for outliers—look for points that are far away from the others.

- The relationship is **linear:** “Linear” means that the relationship between the two variables can be described reasonably well by a straight line. You can use a scatterplot to check whether the relationship between two variables is linear.

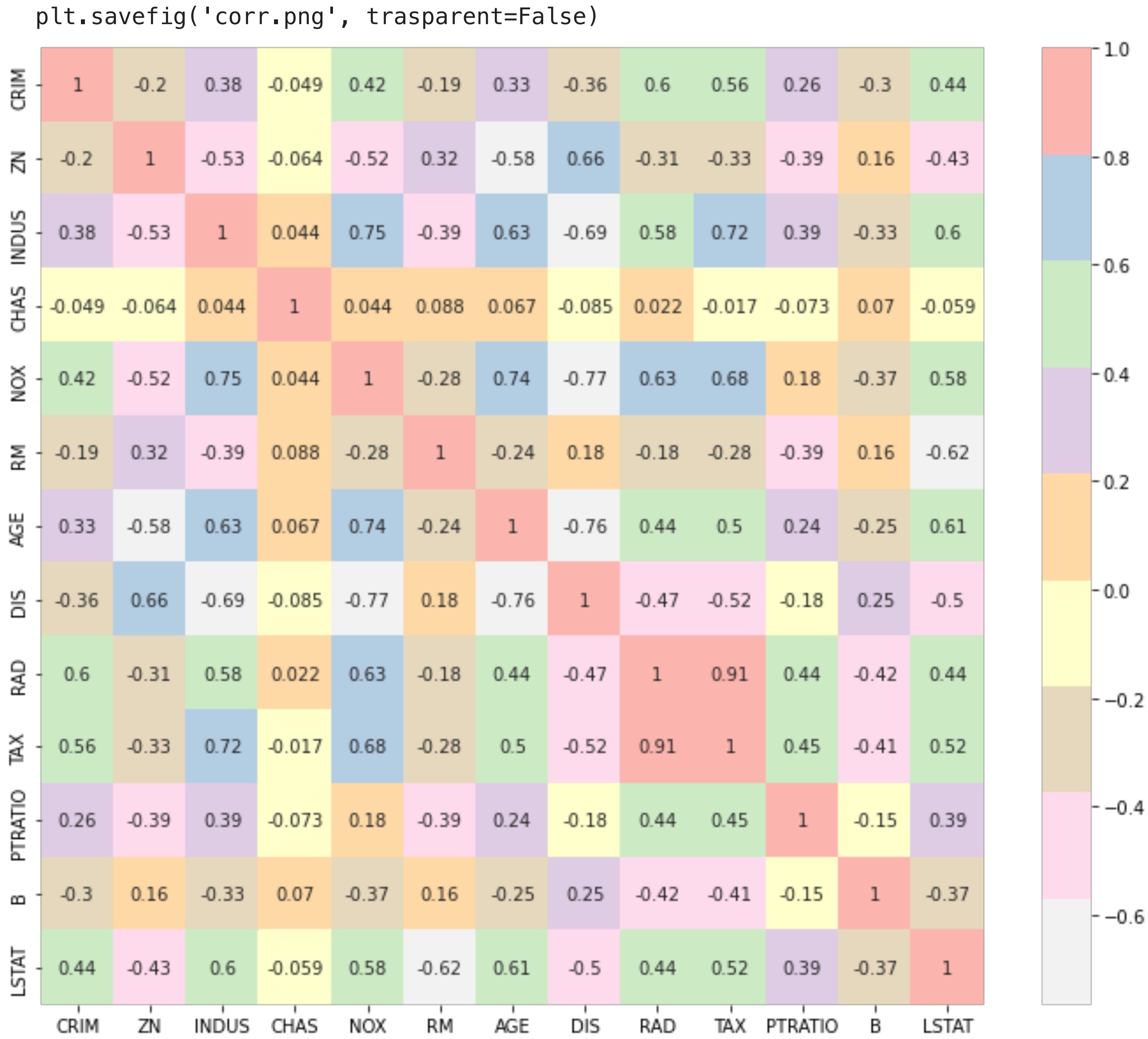
In [8]:
```python
X_train.corr() # finding Realation between every columns. Ignore Diagnols
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATI
CRIM	1.000000	-0.196172	0.382073	-0.049364	0.416560	-0.188280	0.329927	-0.355840	0.603880	0.560570	0.26478
ZN	-0.196172	1.000000	-0.529392	-0.063863	-0.523572	0.319260	-0.583885	0.658331	-0.314833	-0.327834	-0.39283
INDUS	0.382073	-0.529392	1.000000	0.044224	0.750218	-0.392969	0.629257	-0.686848	0.578459	0.719038	0.38835
CHAS	-0.049364	-0.063863	0.044224	1.000000	0.043748	0.088125	0.067269	-0.085492	0.022338	-0.017156	-0.07268
NOX	0.416560	-0.523572	0.750218	0.043748	1.000000	-0.279202	0.740052	-0.765753	0.627188	0.683445	0.17904
RM	-0.188280	0.319260	-0.392969	0.088125	-0.279202	1.000000	-0.235839	0.183857	-0.179242	-0.275242	-0.38552
AGE	0.329927	-0.583885	0.629257	0.067269	0.740052	-0.235839	1.000000	-0.761543	0.440578	0.502429	0.23972
DIS	-0.355840	0.658331	-0.686848	-0.085492	-0.765753	0.183857	-0.761543	1.000000	-0.467653	-0.519643	-0.17662
RAD	0.603880	-0.314833	0.578459	0.022338	0.627188	-0.179242	0.440578	-0.467653	1.000000	0.907455	0.43768
TAX	0.560570	-0.327834	0.719038	-0.017156	0.683445	-0.275242	0.502429	-0.519643	0.907455	1.000000	0.44751
PTRATIO	0.264780	-0.392838	0.388353	-0.072683	0.179046	-0.385526	0.239729	-0.176620	0.437687	0.447518	1.00000
B	-0.299525	0.164641	-0.331638	0.069682	-0.369445	0.157459	-0.250416	0.248376	-0.415325	-0.412145	-0.14563
LSTAT	0.439369	-0.429178	0.603374	-0.059060	0.577154	-0.623920	0.606530	-0.501780	0.442783	0.515905	0.38775

In [15]:
```python
import seaborn as sns
#Using Pearson Correlation
plt.figure(figsize=(12,10))
cor = X_train.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Pastel1_r)
#plt.savefig('corr.png', trasparent=False)
plt.show()
```

```
/var/folders/0b/_lhktjtd7_nbl39rvt0kjvmr0000gn/T/ipykernel_82545/2019311858.py:6: MatplotlibDeprecationWarni
ng: savefig() got unexpected keyword argument "trasparent" which is no longer supported as of 3.3 and will b
ecome an error two minor releases later
  plt.savefig('corr.png', trasparent=False)
```



In [10]:
```python
def Core(dataset, threshold):
    col_corr = set()  # Using set to store all the names of correlated columns for avoiding col duplications
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

As the threshold for selecting variables, we need to set an absolute value, say 0.6.If we discover that the predictor variables are correlated, we can remove the variable with the lowest correlation coefficient value with the target variable. We can also calculate multiple correlation coefficients to see if more than two variables are related. This is referred to as multicollinearity. In this step we will be removing the features which are highly correlated

Applying the Function **Core** with a threshold of **0.6**

In [45]:
```
corr_features = Core(X_train, 0.6)
len(set(corr_features))
```

Out[45]: 6

In [47]:
```
X_train[corr_features].corr()
```

```
/var/folders/0b/_lhktjtd7_nbl39rvt0kjvmr0000gn/T/ipykernel_82262/2874784839.py:1: FutureWarning: Passing a s
et as an indexer is deprecated and will raise in a future version. Use a list instead.
  X_train[corr_features].corr()
```

Out[47]:

	AGE	TAX	RAD	DIS	NOX	LSTAT
AGE	1.000000	0.502429	0.440578	-0.761543	0.740052	0.606530
TAX	0.502429	1.000000	0.907455	-0.519643	0.683445	0.515905
RAD	0.440578	0.907455	1.000000	-0.467653	0.627188	0.442783
DIS	-0.761543	-0.519643	-0.467653	1.000000	-0.765753	-0.501780
NOX	0.740052	0.683445	0.627188	-0.765753	1.000000	0.577154
LSTAT	0.606530	0.515905	0.442783	-0.501780	0.577154	1.000000

In [14]:
```
X_train.drop(corr_features,axis=1)
X_test.drop(corr_features,axis=1)
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	RM	PTRATIO	B
329	0.06724	0.0	3.24	0.0	6.333	16.9	375.21
371	9.23230	0.0	18.10	0.0	6.216	20.2	366.15
219	0.11425	0.0	13.89	1.0	6.373	16.4	393.74
403	24.80170	0.0	18.10	0.0	5.349	20.2	396.90
78	0.05646	0.0	12.83	0.0	6.232	18.7	386.40
...	...	...	...	...	...	...	...
4	0.06905	0.0	2.18	0.0	7.147	18.7	396.90
428	7.36711	0.0	18.10	0.0	6.193	20.2	96.73
385	16.81180	0.0	18.10	0.0	5.277	20.2	396.90
308	0.49298	0.0	9.90	0.0	6.635	18.4	396.90
5	0.02985	0.0	2.18	0.0	6.430	18.7	394.12

152 rows × 7 columns

Performing Correlation Test on Santander

We are going to use the Variance pass dataset here for the correlation test

In [18]:
```
df=pd.read_csv('train.csv',nrows=10000)
X=df.drop(labels=['TARGET'], axis=1)
y=df['TARGET']
# separate dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

In [30]:
```
# Loading the VT Pass data
data = pd.read_csv('var-pass.csv')
data.sample(3)
```

Out[30]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_op_var40_comer_ult1	imp_
6637	2714	2	33	0.0	0.0	0.0	0.0	
4897	4975	2	40	1200.0	0.0	0.0	0.0	
6015	2100	2	23	0.0	0.0	0.0	0.0	

3 rows × 284 columns

In [33]:
```
corr_features = Core(data, 0.9)
len(set(corr_features))
```

Out[33]: 156

Great Job!

That's all we need to know for now! Congratulations, you have learnt one more topic and hands-on with Python.This Notebook is prepared by Sumit Kumar Shukla IBM ICE.

About Author

Mr. Sumit is a Subject Matter Expert at IBM, and a data Scientist with more than five years of experience tutoring students from IITs, NITs, IISc, IIMs, and other prestigious institutions. Google Data Studio certified and IBM certified data analyst Data Science, Machine Learning Models, Graph Databases, and Data Mining techniques for Predictive Modeling and Analytics, as well as data integration, require expertise in Machine Learning and programming languages such as Python, R, and Tableau.

In [ ]: