Data Feature Selection

What is Feature Selection?

Feature Scaling is one of the core concepts in machine learning which hugely impacts the performance of your model. The data features used to train the model have a great influence on the performance you achieve. Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. The performance of machine learning model is directly proportional to the data features used to train it. The performance of ML model will be affected negatively if the data features provided to it are irrelevant. On the other hand, use of relevant data features can increase the accuracy of your ML model especially linear and logistic regression.

Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

How to select features and what are Benefits of performing feature selection before modeling your data?

- **Reduces Overfitting**: Less redundant data means less opportunity to make decisions based on noise.
- **Improves Accuracy**: Less misleading data means modeling accuracy improves.
- **Reduces Training Time**: fewer data points reduce algorithm complexity and algorithms train faster.

Feature Selection Methods:

1. Univariate Selection
2. Feature Importance
3. Correlation Matrix with Heatmap

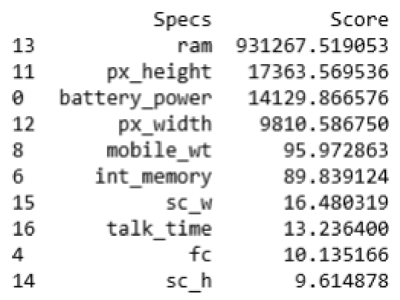
Univariate Selection

Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the **SelectKBest** class that can be used with a suite of different statistical tests to select a specific number of features. The example below uses the chi-squared (chi²) statistical test for non-negative features to select 10 of the best features from the Mobile Price Range Prediction Dataset.

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
data = pd.read_csv("D://Blogs//train.csv")
X = data.iloc[:,0:20]
y = data.iloc[:,-1]
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
```

```
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']
print(featureScores.nlargest(10,'Score'))
```

Output:



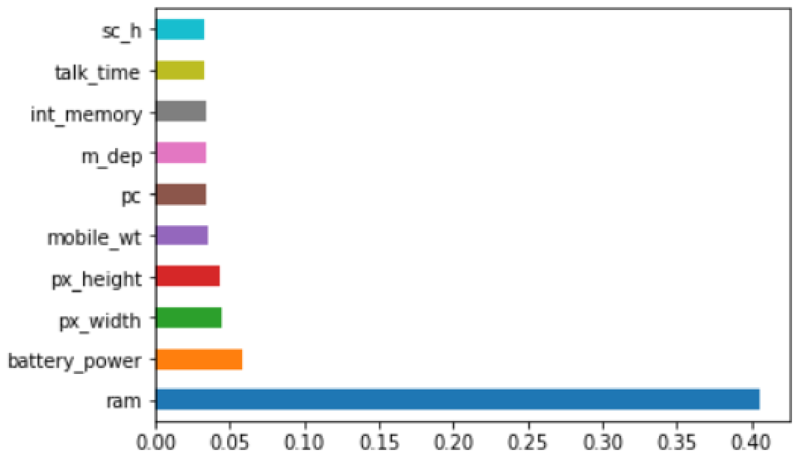
```
            Specs          Score
13            ram  931267.519053
11      px_height   17363.569536
0   battery_power   14129.866576
12       px_width    9810.586750
8       mobile_wt      95.972863
6      int_memory      89.839124
15           sc_w      16.480319
16      talk_time      13.236400
4              fc      10.135166
14           sc_h       9.614878
```

Feature Importance

You can get the feature importance of each feature of your dataset by using the feature importance property of the model. Feature importance gives you a score for each feature of your data, the higher the score more important or relevant is the feature towards your output variable. Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top ten features for the dataset.

```
import pandas as pd
import numpy as np
data = pd.read_csv("D://Blogs//train.csv")
X = data.iloc[:,0:20]
y = data.iloc[:,-1]
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model = ExtraTreesClassifier()
model.fit(X,y)
print(model.feature_importances_)
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

Output:



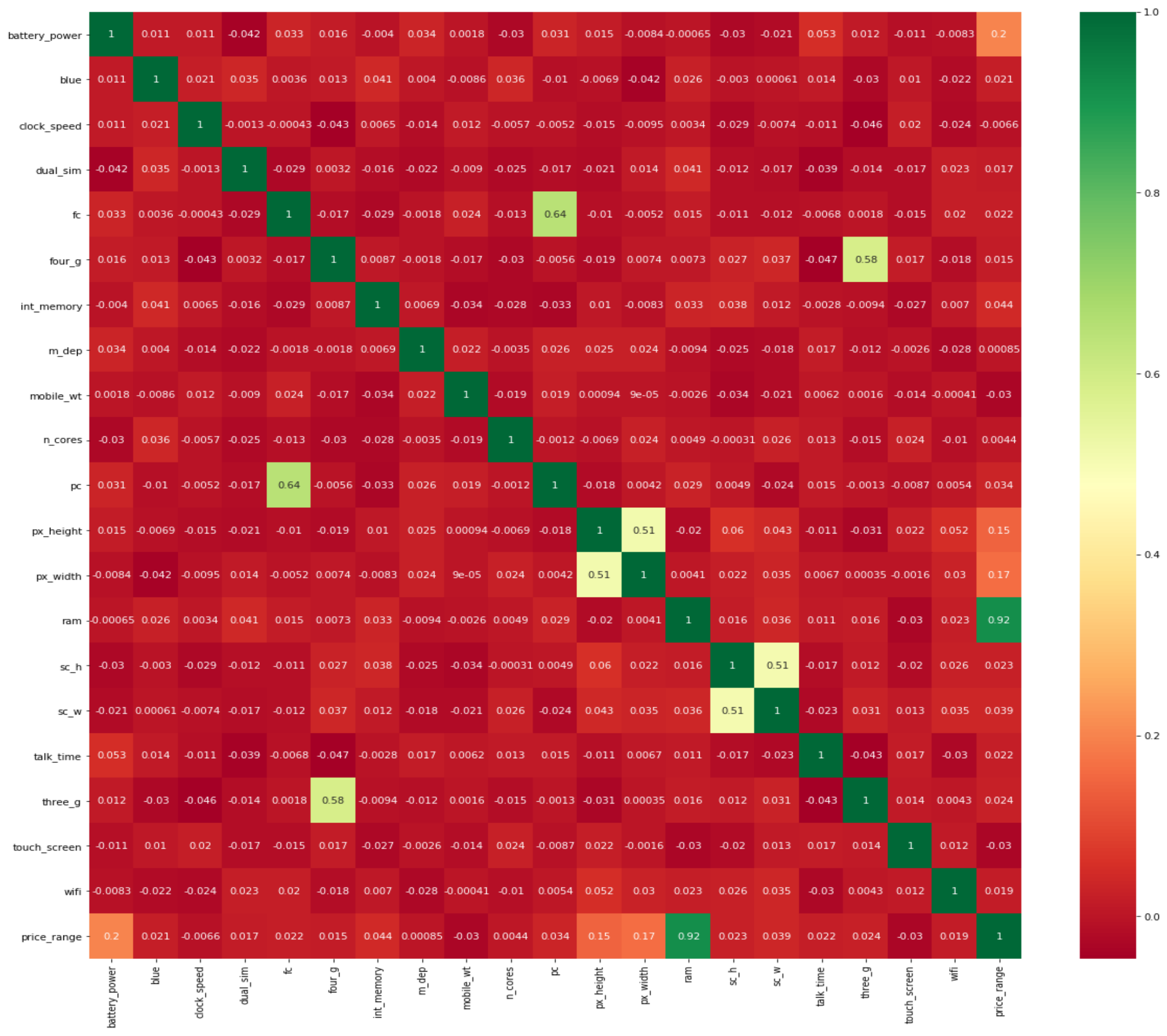
top 10 most important features in data

Correlation Matrix with Heatmap

Correlation states how the features are related to each other or the target variable. Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable). Heatmap makes it easy to identify which features are most related to the target variable, we will plot heatmap of correlated features using the seaborn library.

```
import pandas as pd
import numpy as np
import seaborn as sns
data = pd.read_csv("D://Blogs//train.csv")
X = data.iloc[:,0:20]
y = data.iloc[:,-1]
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

Output:



Have a look at the last row i.e price range, see how the price range is correlated with other features, ram is the highly correlated with price range followed by battery power, pixel height and width while m_dep, clock_speed and n_cores seems to be least correlated with price_range.

Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation which converts a set of correlated variables to a set of uncorrelated variables. PCA is a most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover, PCA is an unsupervised statistical technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.

Let's take an example to perform Principal Component Analysis on the Breast cancer dataset.

Let's start by importing the libraries:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

Now let's load the dataset:

```
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
```

```
cancer.keys()
```

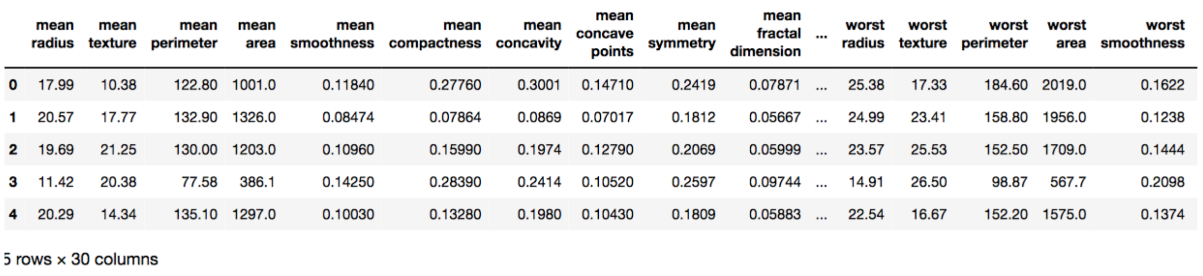
```
dict_keys(['DESCR', 'data', 'feature_names', 'target_names', 'target'])
```

Creating dataframe:

```
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
```

```
df.head()
```

Output:



	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374

5 rows × 30 columns

Now, further we will be importing the StandardScaler class.

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaler.fit(df)
```

```
scaled_data = scaler.transform(df)    #fitting
```

```
# Importing PCA
from sklearn.decomposition import PCA
```

```
# Let's say, components = 2
pca = PCA(n_components=2)
```

```
pca.fit(scaled_data)
```

Now we can transform this data to its first 2 principal components.

```
x_pca = pca.transform(scaled_data)
```

```
scaled_data.shape
```

```
(569, 30)
```

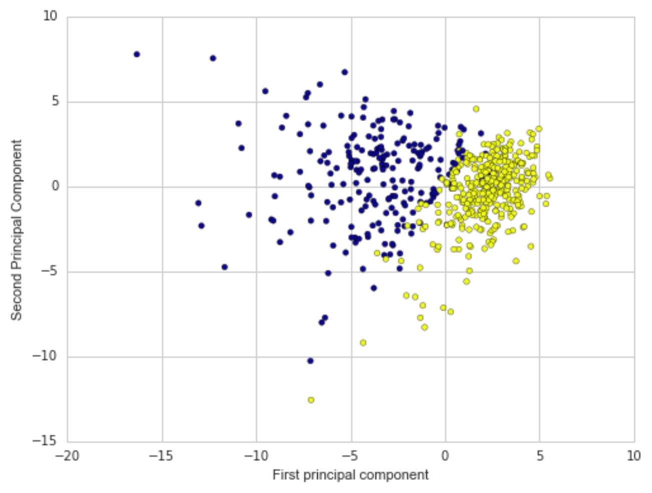
```
x_pca.shape
```

```
(569, 2)
```

Now, Let's plot these two dimensions out!

```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='plasma')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
```

```
<matplotlib.text.Text at 0x11eb56908>
```



```
#pca components
pca.components_
```

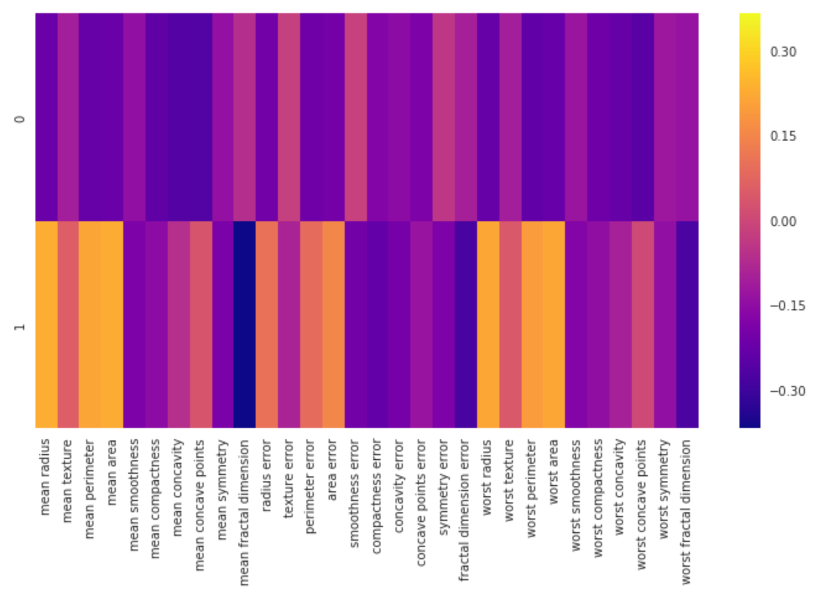
```
array([[-0.21890244, -0.10372458, -0.22753729, -0.22099499, -0.14258969,
        -0.23928535, -0.25840048, -0.26085376, -0.13816696, -0.06436335,
        -0.20597878, -0.01742803, -0.21132592, -0.20286964, -0.01453145,
        -0.17039345, -0.15358979, -0.1834174 , -0.04249842, -0.10256832,
        -0.22799663, -0.10446933, -0.23663968, -0.22487053, -0.12795256,
        -0.21009588, -0.22876753, -0.25088597, -0.12290456, -0.13178394],
       [ 0.23385713,  0.05970609,  0.21518136,  0.23107671, -0.18611302,
        -0.15189161, -0.06016536,  0.0347675 , -0.19034877, -0.36657547,
         0.10555215, -0.08997968,  0.08945723,  0.15229263, -0.20443045,
        -0.2327159 , -0.19720728, -0.13032156, -0.183848  , -0.28009203,
         0.21986638,  0.0454673 ,  0.19987843,  0.21935186, -0.17230435,
        -0.14359317, -0.09796411,  0.00825724, -0.14188335, -0.27533947]])
```

In this numpy matrix array, each row represents a principal component, and each column relates back to the original features. we can visualize this relationship with a heatmap:

```
df_comp = pd.DataFrame(pca.components_,columns=cancer['feature_names'])
```

```
plt.figure(figsize=(12,6))
sns.heatmap(df_comp,cmap='plasma',)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11d546f98>
```


mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension