Step by Step process of Feature Engineering for Machine Learning Algorithms in Data Science

MACHINE LEARNING PYTHON STRUCTURED DATA SUPERVISED

Introduction

Data Science is not a field where theoretical understanding helps you to start a carrier. It totally depends on the projects you do and the practice you have done that determines your probability of success. Feature engineering is a very important aspect of machine learning and data science and should never be ignored. The main goal of Feature engineering is to get the best results from the algorithms.

Table of Contents

1. Why should we use Feature Engineering in data science?

In Data Science, the performance of the model is depending on data preprocessing and data handling. Suppose if we build a model without Handling data, we got an accuracy of around 70%. By applying the Feature engineering on the same model there is a chance to increase the performance from 70% to more.

Simply, by using Feature Engineering we improve the performance of the model.

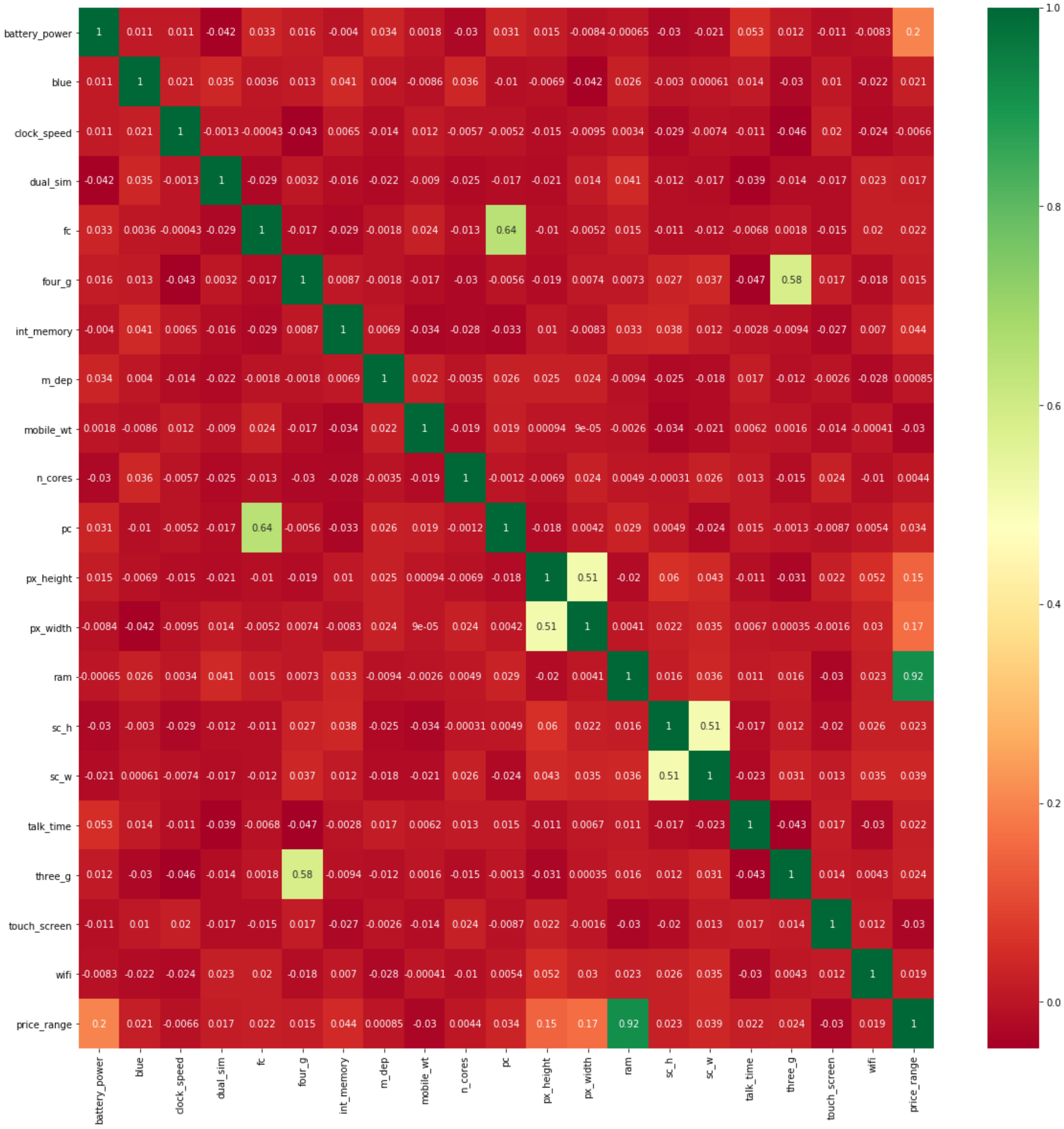
2. Feature selection

Feature selection is nothing but a selection of required independent features. Selecting the important independent features which have more relation with the dependent feature will help to build a good model. There are some methods for feature selection:

2.1 Correlation Matrix with Heatmap

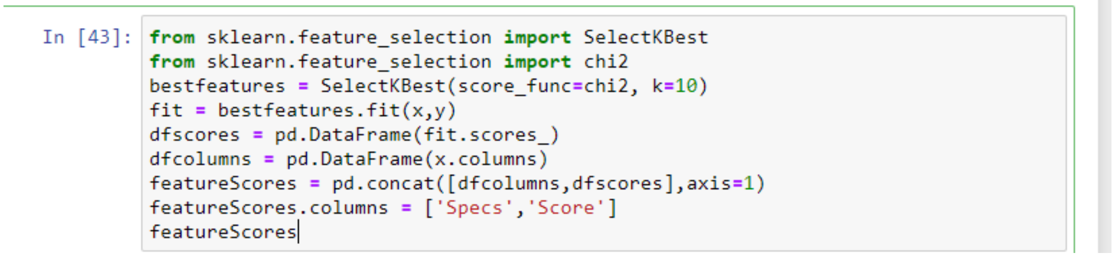
Heatmap is a graphical representation of 2D (two-dimensional) data. Each data value represents in a matrix.

Firstly, plot the pair plot between all independent features and dependent features. It will give the relation between dependent and independent features. The relation between the independent feature and the dependent feature is less than 0.2 then choose that independent features for building a model.

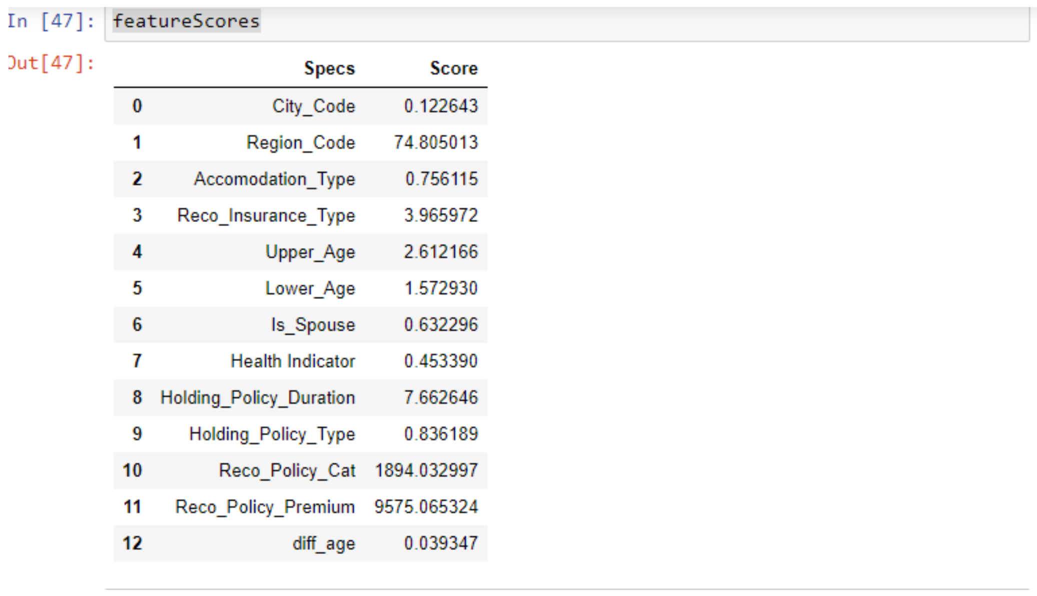


2.2 Univariate Selection

In this, Statistical tests can be used to select the independent features which have the strongest relationship with the dependent feature. SelectKBest method can be used with a suite of different statistical tests to select a specific number of features.



```
In [43]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import chi2
         bestfeatures = SelectKBest(score_func=chi2, k=10)
         fit = bestfeatures.fit(x,y)
         dfscores = pd.DataFrame(fit.scores_)
         dfcolumns = pd.DataFrame(x.columns)
         featureScores = pd.concat([dfcolumns,dfscores],axis=1)
         featureScores.columns = ['Specs','Score']
         featureScores
```



```
In [47]: featureScores
```

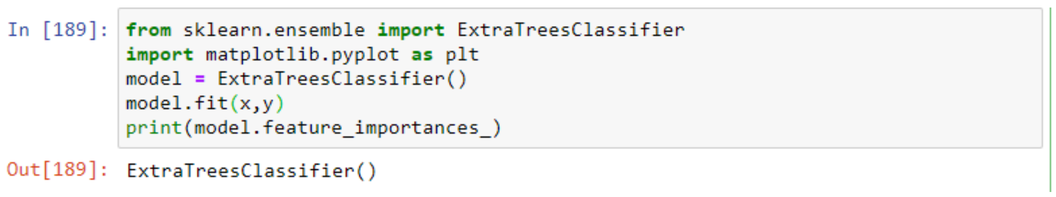
Out[47]:

	Specs	Score
0	City_Code	0.122643
1	Region_Code	74.805013
2	Accomodation_Type	0.756115
3	Reco_Insurance_Type	3.965972
4	Upper_Age	2.612166
5	Lower_Age	1.572930
6	Is_Spouse	0.632296
7	Health Indicator	0.453390
8	Holding_Policy_Duration	7.662646
9	Holding_Policy_Type	0.836189
10	Reco_Policy_Cat	1894.032997
11	Reco_Policy_Premium	9575.065324
12	diff_age	0.039347

- Which feature has the highest score will be more related to the dependent feature and choose those features for the model.

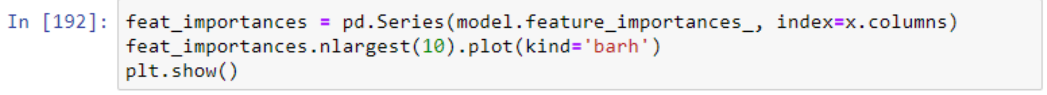
2.3 ExtraTreesClassifier method

In this method, the ExtraTreesClassifier method will help to give the importance of each independent feature with a dependent feature. Feature importance will give you a score for each feature of your data, the higher the score more important or relevant to the feature towards your output variable.

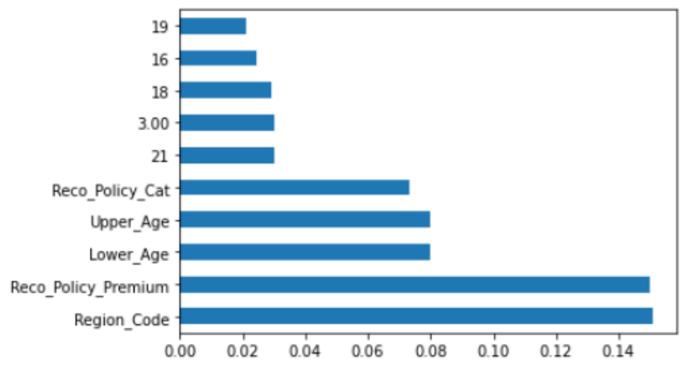


```
In [189]: from sklearn.ensemble import ExtraTreesClassifier
          import matplotlib.pyplot as plt
          model = ExtraTreesClassifier()
          model.fit(x,y)
          print(model.feature_importances_)

Out[189]: ExtraTreesClassifier()
```



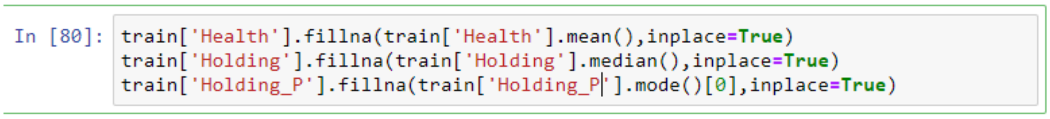
```
In [192]: feat_importances = pd.Series(model.feature_importances_, index=x.columns)
          feat_importances.nlargest(10).plot(kind='barh')
          plt.show()
```



3. Handling Missing Values

In some datasets, we got the NA values in features. It is nothing but missing data. By handling this type of data there are many ways:

- In the missing value places, to replace the missing values with mean or median to numerical data and for categorical data with mode.

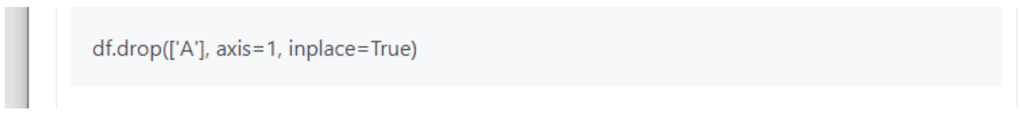


```
In [80]: train['Health'].fillna(train['Health'].mean(),inplace=True)
         train['Holding'].fillna(train['Holding'].median(),inplace=True)
         train['Holding_P'].fillna(train['Holding_P'].mode()[0],inplace=True)
```

- Drop NA values entire rows.

```
df.dropna(how = 'all')
```

- Drop NA values entire features. (it helps if NA values are more than 50% in a feature)



```
df.drop(['A'], axis=1, inplace=True)
```

- Replace NA values with 0.

```
df_result = df.fillna(0)
```

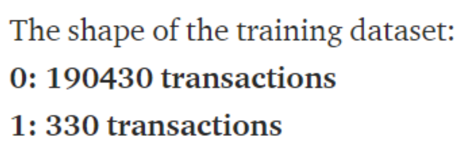
If you choose to drop options, there is a chance to lose important information from them. So better to choose to replace options.

4. Handling imbalanced data

Why need to handle imbalanced data? Because of to reduce overfitting and underfitting problem.

suppose a feature has a factor level2(0 and 1). it consists of 1's is 5% and 0's is 95%. It is called imbalanced data.

Example:-



The shape of the training dataset:

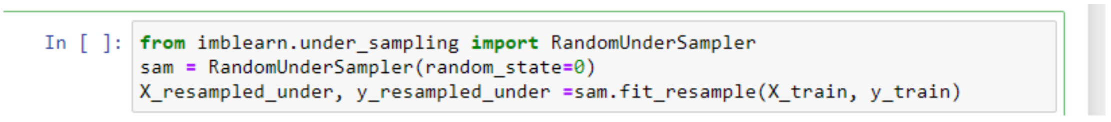
0: 190430 transactions

1: 330 transactions

By preventing this problem there are some methods:

4.1 Under-sampling majority class

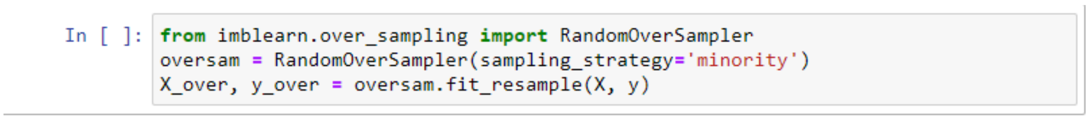
Under-sampling the majority class will resample the majority class points in the data to make them equal to the minority class.



```
In [ ]: from imblearn.under_sampling import RandomUnderSampler
        sam = RandomUnderSampler(random_state=0)
        X_resampled_under, y_resampled_under =sam.fit_resample(X_train, y_train)
```

4.2 Over Sampling Minority class by duplication

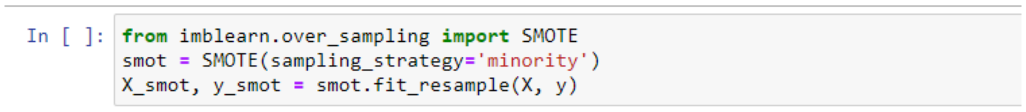
Oversampling minority class will resample the minority class points in the data to make them equal to the majority class.



```
In [ ]: from imblearn.over_sampling import RandomOverSampler
        oversam = RandomOverSampler(sampling_strategy='minority')
        X_over, y_over = oversam.fit_resample(X, y)
```

4.3 Over Sampling minority class using Synthetic Minority Oversampling Technique (SMOTE)

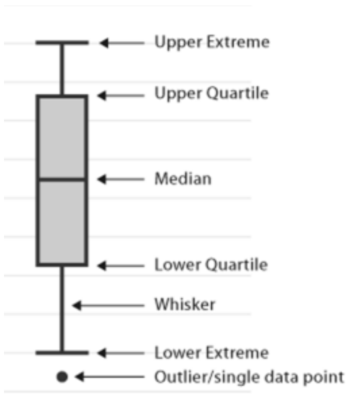
In this method, synthetic samples are generated for the minority class and equal to the majority class.



```
In [ ]: from imblearn.over_sampling import SMOTE
        smot = SMOTE(sampling_strategy='minority')
        X_smot, y_smot = smot.fit_resample(X, y)
```

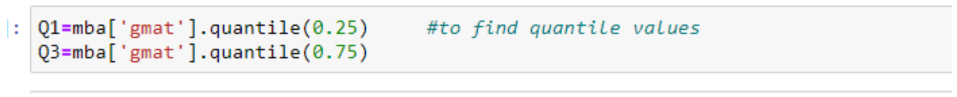
5. Handling outliers

firstly, calculate the skewness of the features and check whether they are positively skewed, negatively skewed, or normally skewed. Another method is to plot the boxplot to features and check if any values are out of bounds or not. if there, they are called outliers.



how to handle these outliers: –

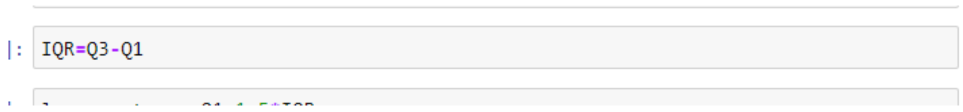
first, calculate quantile values at 25% and 75%.



```
|: Q1=mba['gmat'].quantile(0.25)     #to find quantile values
   Q3=mba['gmat'].quantile(0.75)
```

- next, calculate the Interquartile range

IQR = Q3 – Q1

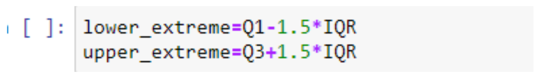


```
|: IQR=Q3-Q1
```

- next, calculate the upper extreme and lower extreme values

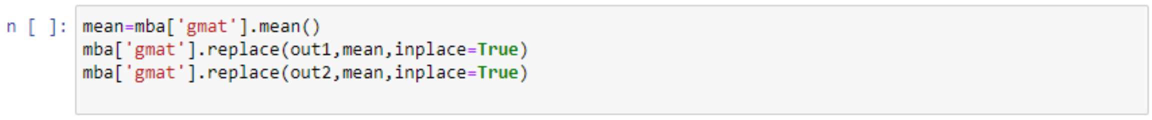
lower extreme=Q1 – 1.5 * IQR

upper extreme=Q3– 1.5 * IQR



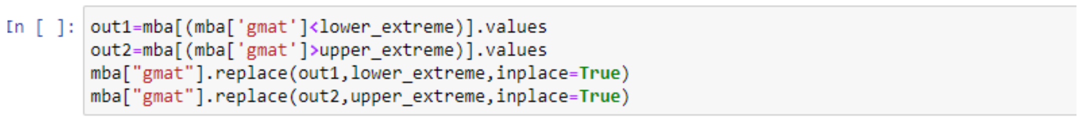
```
[ ]: lower_extreme=Q1-1.5*IQR
     upper_extreme=Q3+1.5*IQR
```

- lastly, check the values will lie above the upper extreme or below the lower extreme. if it presents then remove them or replace them with mean, median, or any quantile values.
- Replace outliers with mean



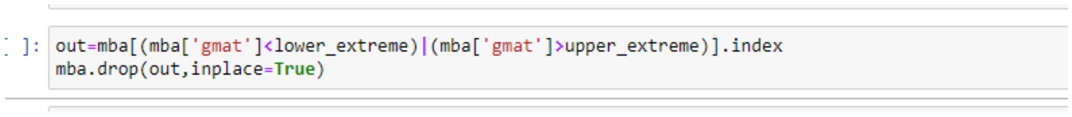
```
n [ ]: mean=mba['gmat'].mean()
       mba['gmat'].replace(out1,mean,inplace=True)
       mba['gmat'].replace(out2,mean,inplace=True)
```

- Replace outliers with quantile values



```
In [ ]: out1=mba[(mba['gmat']<lower_extreme)].values
        out2=mba[(mba['gmat']>upper_extreme)].values
        mba["gmat"].replace(out1,lower_extreme,inplace=True)
        mba["gmat"].replace(out2,upper_extreme,inplace=True)
```

- Drop outliers

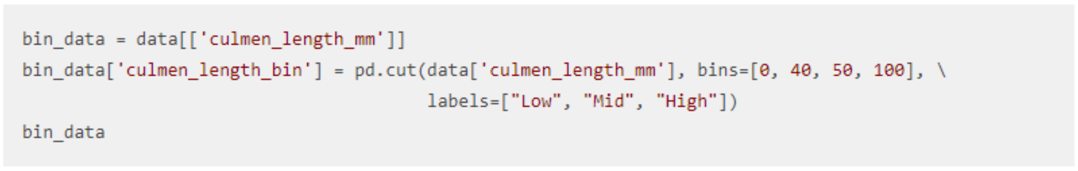


```
 ]: out=mba[(mba['gmat']<lower_extreme)|(mba['gmat']>upper_extreme)].index
    mba.drop(out,inplace=True)
```

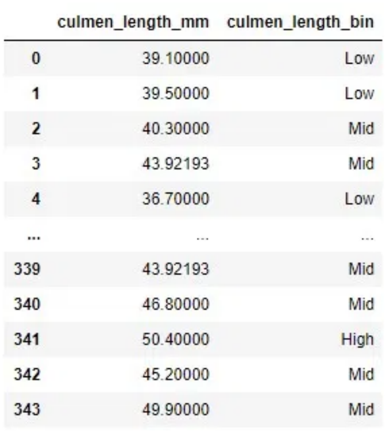
6. Binning

Binning is nothing but any data value within the range is made to fit into the bin. It is important in your data exploration activity. We typically use it to transform continuous variables into discrete ones.

Suppose if we have AGE feature in continuous and we need to divide the age into groups as a feature then it will be useful.



```
bin_data = data[['culmen_length_mm']]
bin_data['culmen_length_bin'] = pd.cut(data['culmen_length_mm'], bins=[0, 40, 50, 100], \
                                        labels=["Low", "Mid", "High"])
bin_data
```

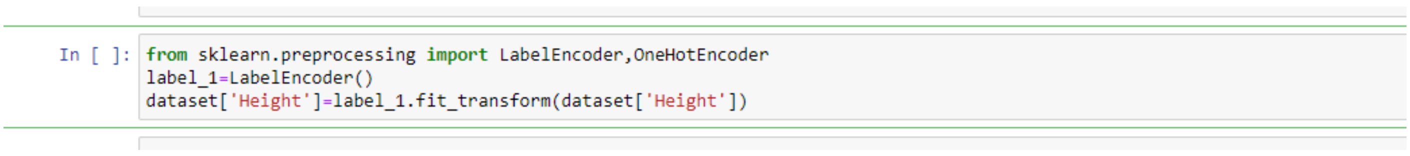


	culmen_length_mm	culmen_length_bin
0	39.10000	Low
1	39.50000	Low
2	40.30000	Mid
3	43.92193	Mid
4	36.70000	Low
...	...	...
339	43.92193	Mid
340	46.80000	Mid
341	50.40000	High
342	45.20000	Mid
343	49.90000	Mid

7. Encoding:

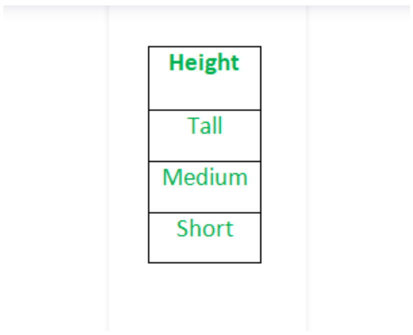
Why this will apply? because in datasets we may contain object datatypes. for building a model we need to have all features are in integer datatypes. so, Label Encoder and OneHotEncoder are used to convert object datatype to integer datatype.

- Label Encoding



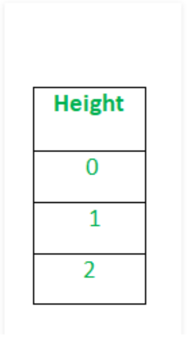
```
In [ ]: from sklearn.preprocessing import LabelEncoder,OneHotEncoder
        label_1=LabelEncoder()
        dataset['Height']=label_1.fit_transform(dataset['Height'])
```

Before applying Label Encoding



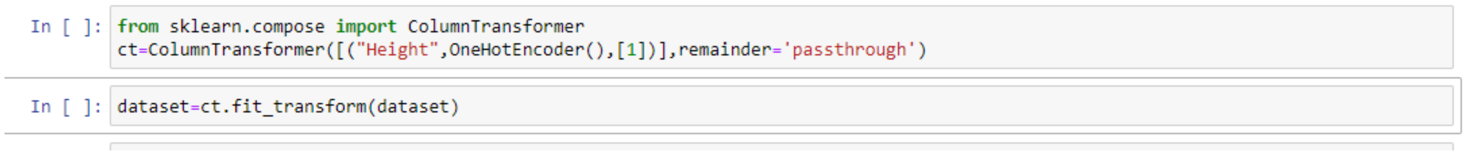
Height
Tall
Medium
Short

After applying label encoding, the Height column is converted into:



Height
0
1
2

After applying label encoding then apply the column transformer method to convert labels to 0 and 1

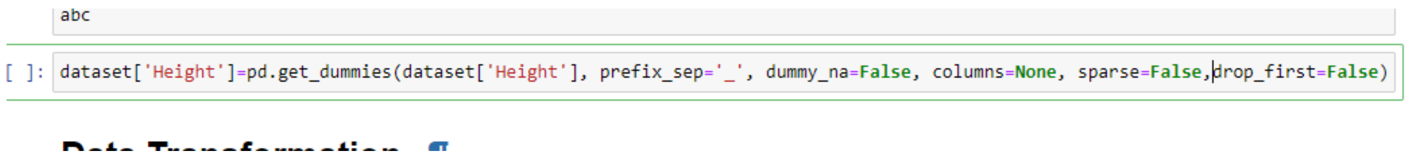


```
In [ ]: from sklearn.compose import ColumnTransformer
        ct=ColumnTransformer([("Height",OneHotEncoder(),[1])],remainder='passthrough')

In [ ]: dataset=ct.fit_transform(dataset)
```

- One Hot Encoding:

By applying get_dummies we convert directly categorical to numerical



```
abc

[ ]: dataset['Height']=pd.get_dummies(dataset['Height'], prefix_sep='_', dummy_na=False, columns=None, sparse=False,drop_first=False)
```

8. Feature scaling

Why this scaling is applying? because to reduce the variance effect and to overcome the fitting problem. there are two types of scaling methods:

8.1 Standardization

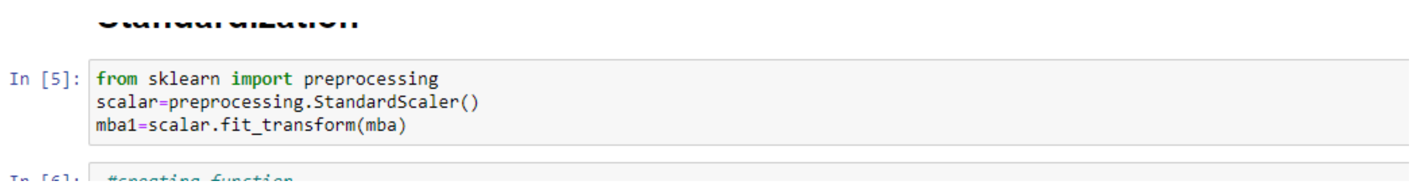
When this method is used?. when all features are having high values, not 0 and 1.

It is a technique to standardize the independent features that present in a fixed range to bring all values to the same magnitudes.

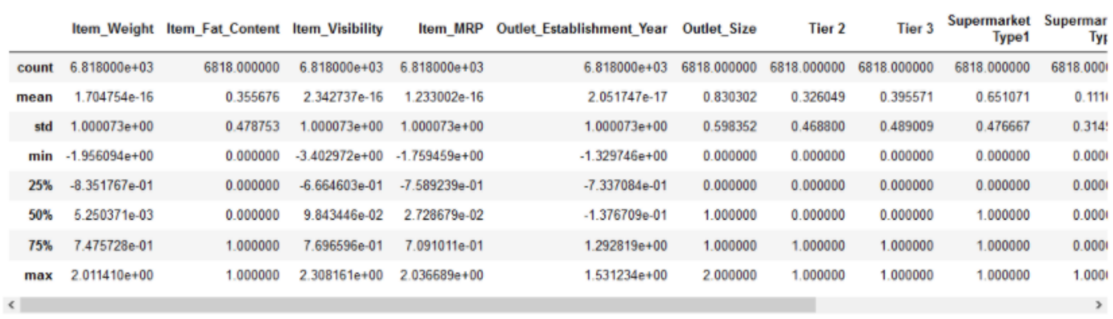
$$Z = \frac{X - \mu}{\sigma}$$

In standardization, the mean of the independent features is 0 and the standard deviation is 1.

Method 1:

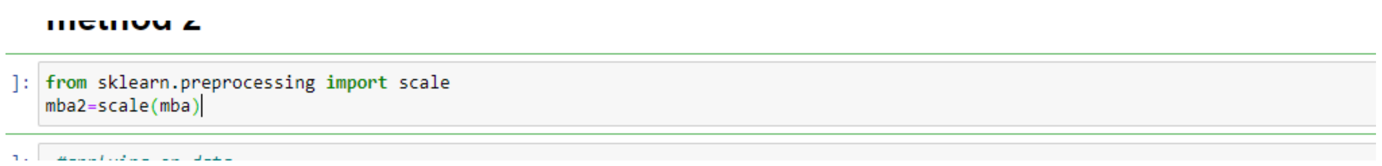


```
In [5]: from sklearn import preprocessing
        scalar=preprocessing.StandardScaler()
        mba1=scalar.fit_transform(mba)
```



	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermar Typ
count	6.818000e+03	6818.000000	6.818000e+03	6.818000e+03	6.818000e+03	6818.000000	6818.000000	6818.000000	6818.000000	6818.000
mean	1.704754e-16	0.355676	2.342737e-16	1.233002e-16	2.051747e-17	0.830302	0.326049	0.395571	0.651071	0.111
std	1.000073e+00	0.478753	1.000073e+00	1.000073e+00	1.000073e+00	0.598352	0.468800	0.489009	0.476667	0.314
min	-1.956094e+00	0.000000	-3.402972e+00	-1.759459e+00	-1.329746e+00	0.000000	0.000000	0.000000	0.000000	0.000
25%	-8.351767e-01	0.000000	-6.664603e-01	-7.589239e-01	-7.337084e-01	0.000000	0.000000	0.000000	0.000000	0.000
50%	5.250371e-03	0.000000	9.843446e-02	2.728679e-02	-1.376709e-01	1.000000	0.000000	0.000000	1.000000	0.000
75%	7.475728e-01	1.000000	7.696596e-01	7.091011e-01	1.292819e+00	1.000000	1.000000	1.000000	1.000000	0.000
max	2.011410e+00	1.000000	2.308161e+00	2.036689e+00	1.531234e+00	2.000000	1.000000	1.000000	1.000000	1.000

Method2:



```
]: from sklearn.preprocessing import scale
   mba2=scale(mba)
```

After encoding feature labels are in 0 and 1. This may affect standardization. To overcome this, we use Normalization.

8.2 Normalisation

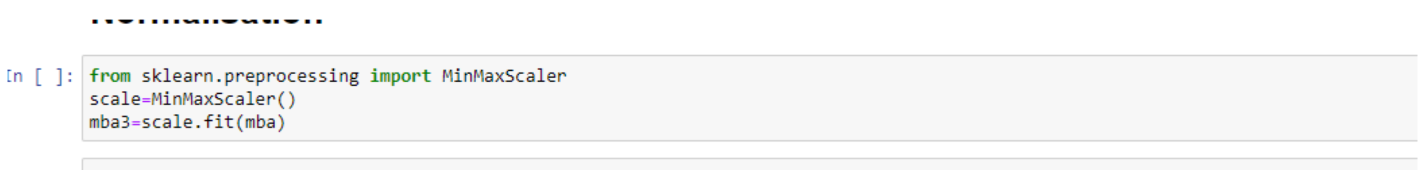
Normalization also makes the training process less sensitive by the scale of the features. This results in getting better coefficients after training.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

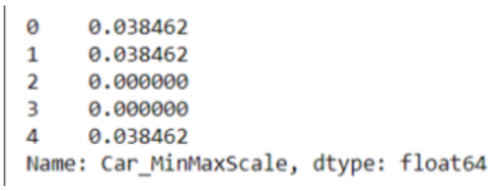
Normalization

Method 1: -MinMaxScaler

It is a method to rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range.



```
In [ ]: from sklearn.preprocessing import MinMaxScaler
        scale=MinMaxScaler()
        mba3=scale.fit(mba)
```

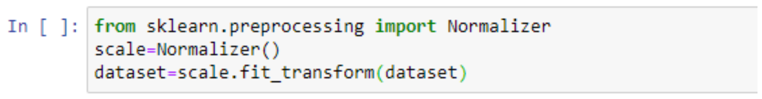


```
0    0.038462
1    0.038462
2    0.000000
3    0.000000
4    0.038462
Name: Car_MinMaxScale, dtype: float64
```

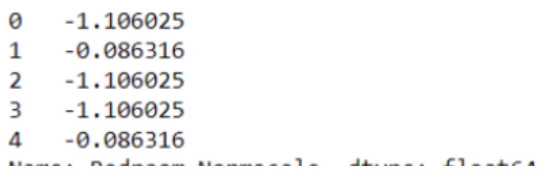
Method 2: – Mean Normalization

It is a method to rescales the feature to a hard and fast range of [-1,1] with mean=0.

$$x' = \frac{x - x_{mean}}{x_{max} - x_{min}}$$



```
In [ ]: from sklearn.preprocessing import Normalizer
        scale=Normalizer()
        dataset=scale.fit_transform(dataset)
```



```
0   -1.106025
1   -0.086316
2   -1.106025
3   -1.106025
4   -0.086316
```

End Notes:-

In this article, I covered step by step process of feature engineering. This is more helpful to increase prediction accuracy.

Keep in mind that there are no particular methods to increase your prediction accuracy. It all depends on your data and applies multiple methods.

As a next step, I encourage you to try out different datasets and analyze them. And don't forget to share your insights in the comments section below!

About the Author:

I am Pavan Kumar Reddy Elluru. I completed my graduation at G.Pullareddy Engineering College in the year 2020. I am a certified data scientist in the year 2021 and passionated about Machine Learning and Deep Learning Projects.

Please ping me in case of any queries or just to say hi!

Email id:- pawankumarreddy1999@gmail.com

Linkedin id: – www.linkedin.com/in/elluru-pavan-kumar-reddy-a1b183197

Github id: – [pawankumarreddy1999 (Pavan Kumar Reddy Elluru) (github.com)](github.com)

Article Url - https://www.analyticsvidhya.com/blog/2021/03/step-by-step-process-of-feature-engineering-for-machine-learning-algorithms-in-data-science/

elluru_pavan_kumar