Practical Guide on Data Preprocessing in Python using Scikit Learn

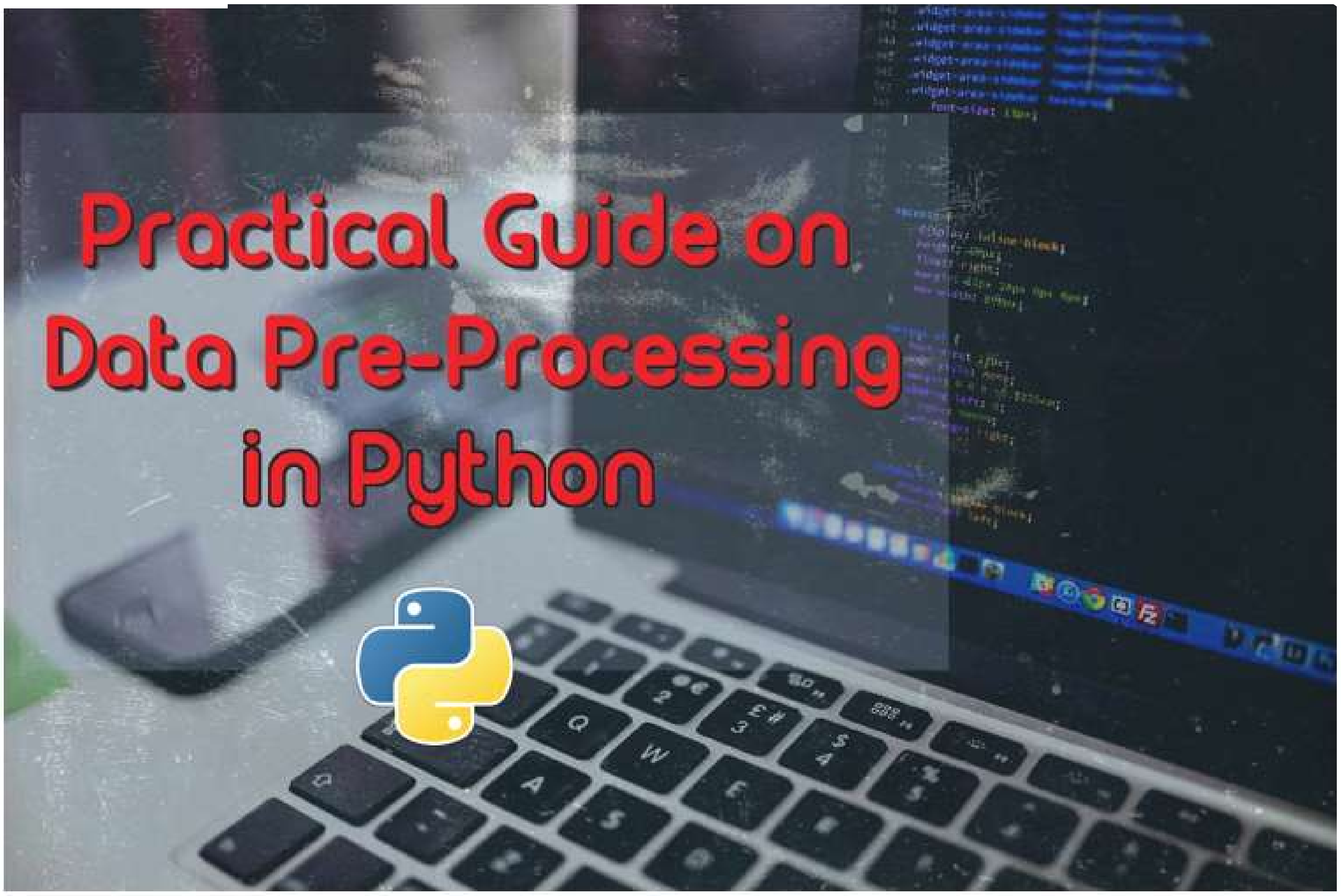
S syed
Last Updated : 21 Oct, 2024

Introduction

This article primarily focuses on data pre-processing techniques in python. Learning algorithms have affinity towards certain data types on which they perform incredibly well. They are also known to give reckless predictions with unscaled or unstandardized features. Algorithm like XGBoost, specifically requires dummy encoded data while algorithm like decision tree doesn't seem to care at all (sometimes)!

In simple words, pre-processing refers to the transformations applied to your data before feeding it to the algorithm. In python, scikit-learn library has a pre-built functionality under sklearn.preprocessing. There are many more options for pre-processing which we'll explore.

After finishing this article, you will be equipped with the basic techniques of data pre-processing and their in-depth understanding. For your convenience, I've attached some resources for in-depth learning of machine learning algorithms and designed few exercises to get a good grip of the concepts.



Available Data set

For this article, I have used a subset of the [Loan Prediction](#) (missing value observations are dropped) data set from You can download the final training and testing data set from here: [Download Data](#)

***Note :** Testing data that you are provided is the subset of the training data from Loan Prediction problem.*

Now, lets get started by importing important packages and the data set.

Copy Code

```
# Importing pandas
>> import pandas as pd
# Importing training data set
>> X_train=pd.read_csv('X_train.csv')
>> Y_train=pd.read_csv('Y_train.csv')
# Importing testing data set
```

```
>> X_test=pd.read_csv('X_test.csv')
>> Y_test=pd.read_csv('Y_test.csv')
```

Lets take a closer look at our data set.

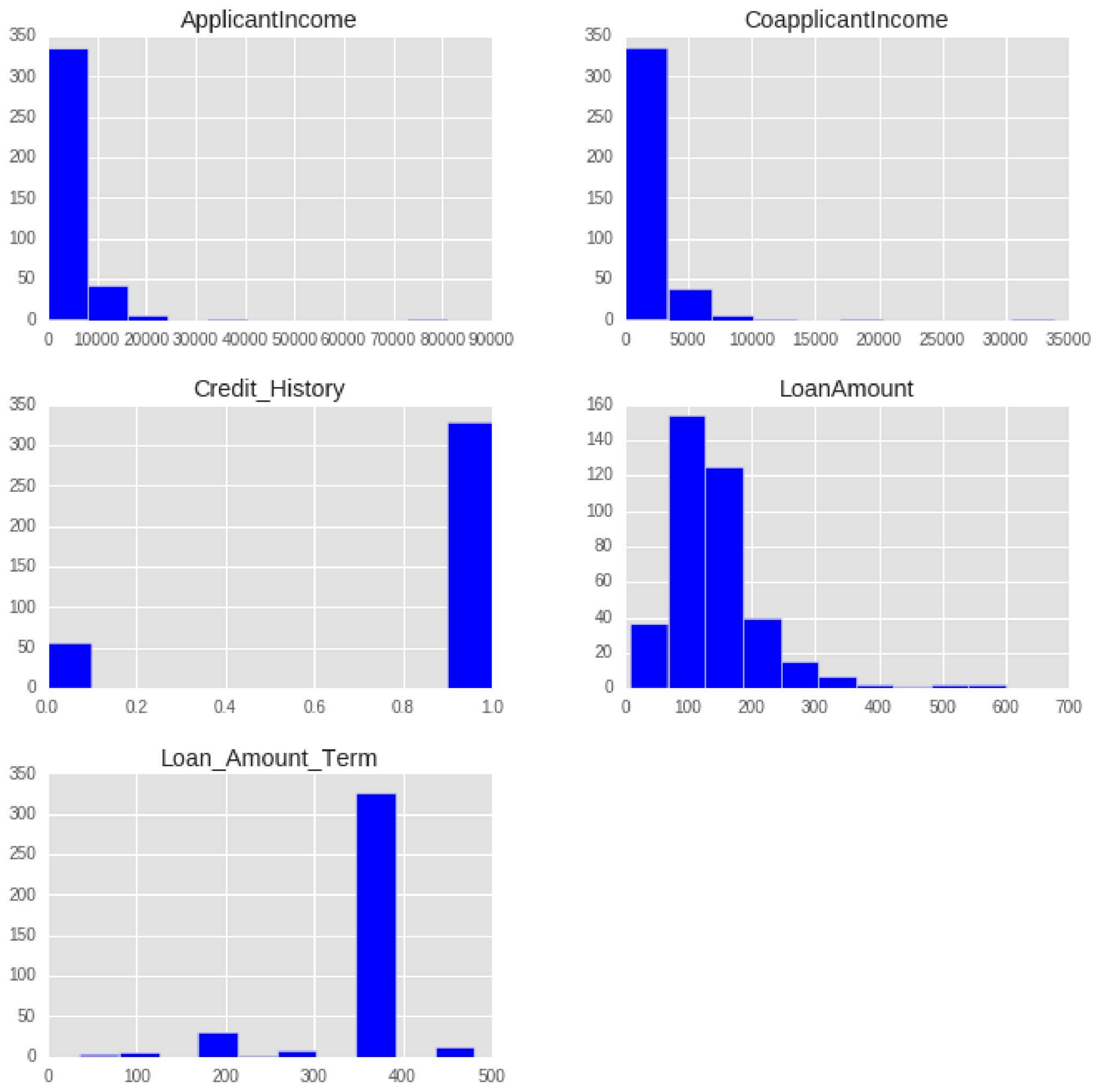
```
>> print (X_train.head())

      Loan_ID Gender Married Dependents Education Self_Employed
15   LP001032   Male      No          0  Graduate            No
248  LP001824   Male     Yes          1  Graduate            No
590  LP002928   Male     Yes          0  Graduate            No
246  LP001814   Male     Yes          2  Graduate            No
388  LP002244   Male     Yes          0  Graduate            No

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term
15              4950                0.0       125.0             360.0
248             2882             1843.0       123.0             480.0
590             3000             3416.0        56.0             180.0
246             9703                0.0       112.0             360.0
388             2333             2417.0       136.0             360.0

     Credit_History Property_Area
15              1.0         Urban
248             1.0     Semiurban
590             1.0     Semiurban
246             1.0         Urban
388             1.0         Urban
```

Feature Scaling

Feature scaling is the method to limit the range of variables so that they can be compared on common grounds. It is performed on continuous variables. Lets plot the distribution of all the continuous variables in the data set.

Copy Code

```
>> import matplotlib.pyplot as plt
>> X_train[X_train.dtypes[(X_train.dtypes=="float64")|(X_train.dtypes=="int64")]
                         .index.values].hist(figsize=[11,11])
```



After understanding these plots, we infer that `ApplicantIncome` and `CoapplicantIncome` are in similar range (0-50000$) where as `LoanAmount` is in thousands and it ranges from 0 to 600$. The story for `Loan_Amount_Term` is completely different from other variables because its unit is months as opposed to other variables where the unit is dollars.

If we try to apply distance based methods such as kNN on these features, feature with the largest range will dominate the outcome results and we'll obtain less accurate predictions. We can overcome this trouble using feature scaling. Let's do it practically.

Resources : Check out this article on kNN for better understanding.

Wow !! we got an accuracy of 63% just by guessing, What is the meaning of this, getting better accuracy than our prediction model ?

This might be happening because of some insignificant variable with larger range will be dominating the objective function. We can remove this problem by scaling down all the features to a same range. sklearn provides a tool `MinMaxScaler` that will scale down all the features between 0 and 1. Mathematical formula for `MinMaxScaler` is.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Lets try this tool on our problem.

Copy Code

```
# Importing MinMaxScaler and initializing it
>> from sklearn.preprocessing import MinMaxScaler
>> min_max=MinMaxScaler()
# Scaling down both train and test data set
>> X_train_minmax=min_max.fit_transform(X_train[['ApplicantIncome', 'CoapplicantIncome',
                'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])
>> X_test_minmax=min_max.fit_transform(X_test[['ApplicantIncome', 'CoapplicantIncome',
                'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])
```

Now, that we are done with scaling, lets apply kNN on our scaled data and check its accuracy.

Copy Code

```
# Fitting k-NN on our scaled data set
>> knn=KNeighborsClassifier(n_neighbors=5)
>> knn.fit(X_train_minmax,Y_train)
```

```
# Checking the model's accuracy
>> accuracy_score(Y_test,knn.predict(X_test_minmax))
```

```
Out : 0.75
```

Great !! Our accuracy has increased from 61% to 75%. This means that some of the features with larger range were dominating the prediction outcome in the domain of distance based methods(kNN).

It should be kept in mind while performing distance based methods we must attempt to scale the data, so that the feature with lesser significance might not end up dominating the objective function due to its larger range. In addition, features having different unit should also be scaled thus providing each feature equal initial weightage and at the end we will have a better prediction model.

Exercise 1

Try to do the same exercise with a logistic regression model(parameters : penalty='l2',C=0.01) and provide your accuracy before and after scaling in the comment section.

Feature Standardization

Before jumping to this section I suggest you to complete Exercise 1.

In the previous section, we worked on the Loan_Prediction data set and fitted a kNN learner on the data set. After scaling down the data, we have got an accuracy of 75% which is very considerably good. I tried the same exercise on Logistic Regression and I got the following result :

Before Scaling : 61%

After Scaling : 63%

The accuracy we got after scaling is close to the prediction which we made by guessing, which is not a very impressive achievement. So, what is happening here? Why hasn't the accuracy increased by a satisfactory amount as it increased in kNN?

Resources : Go through this article on Logistic Regression for better understanding.

Here is the answer:

In logistic regression, each feature is assigned a weight or coefficient (Wi). If there is a feature with relatively large range and it is insignificant in the objective function then logistic regression will itself assign a very low value to its co-efficient, thus neutralizing the dominant effect of that particular feature, whereas distance based method such as kNN does not have this inbuilt strategy, thus it requires scaling.

Aren't we forgetting something ? Our logistic model is still predicting with an accuracy almost closer to a guess.

Now, I'll be introducing a new concept here called standardization. Many machine learning algorithms in sklearn requires standardized data which means having zero mean and unit variance.

Standardization (or Z-score normalization) is the process where the features are rescaled so that they'll have the properties of a standard normal distribution with μ=0 and σ=1, where μ is the mean (average) and σ is the standard deviation from the mean. Standard scores (also called *z* scores) of the samples are calculated as follows :

$$z = \frac{x - \mu}{\sigma}$$

Elements such as l1 ,l2 regularizer in linear models (logistic comes under this category) and RBF kernel in SVM in objective function of learners assumes that all the features are centered around zero and have variance in the same order.

Features having larger order of variance would dominate on the objective function as it happened in the previous section with the feature having large range. As we saw in the Exercise 1 that without any preprocessing on the data the accuracy was 61%, lets standardize our data apply logistic regression on that. Sklearn provides `scale` to standardize the data.

Copy Code

```
# Standardizing the train and test data
>> from sklearn.preprocessing import scale
>> X_train_scale=scale(X_train[['ApplicantIncome', 'CoapplicantIncome',
                'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])
>> X_test_scale=scale(X_test[['ApplicantIncome', 'CoapplicantIncome',
               'LoanAmount', 'Loan_Amount_Term', 'Credit_History']])
# Fitting logistic regression on our standardized data set
>> from sklearn.linear_model import LogisticRegression
>> log=LogisticRegression(penalty='l2',C=.01)
>> log.fit(X_train_scale,Y_train)
# Checking the model's accuracy
>> accuracy_score(Y_test,log.predict(X_test_scale))
```

Out : `0.75`

We again reached to our maximum score that was attained using kNN after scaling. This means standardizing the data when using a estimator having l1 or l2 regularization helps us to increase the accuracy of the prediction model. Other learners like kNN with euclidean distance measure, k-means, SVM, perceptron, neural networks, linear discriminant analysis, principal component analysis may perform better with standardized data.

Though, I suggest you to understand your data and what kind of algorithm you are going to apply on it; over the time you will be able to judge weather to standardize your data or not.

Note : Choosing between scaling and standardizing is a confusing choice, you have to dive deeper in your data and learner that you are going to use to reach the decision. For starters, you can try both the methods and check cross validation score for making a choice.

Resources : Go through this article on cross validation for better understanding.

Exercise 2

Try to do the same exercise with SVM model and provide your accuracy before and after standardization in the comment section.

Resources : Go through this article on support vector machines for better understanding.

Label Encoding

In previous sections, we did the pre-processing for continuous numeric features. But, our data set has other features too such as `Gender`, `Married`, `Dependents`, `Self_Employed` and `Education`. All these categorical features have string values. For example, `Gender` has two levels either `Male` or `Female`. Lets feed the features in our logistic regression model.

Copy Code

```
# Fitting a logistic regression model on whole data
>> log=LogisticRegression(penalty='l2',C=.01)
>> log.fit(X_train,Y_train)
# Checking the model's accuracy
>> accuracy_score(Y_test,log.predict(X_test))
```

```
Out : ValueError: could not convert string to float: Semiurban
```

We got an error saying that it cannot convert string to float. So, what's actually happening here is learners like logistic regression, distance based methods such as kNN, support vector machines, tree based methods etc. in sklearn needs numeric arrays. Features having string values cannot be handled by these learners.

Sklearn provides a very efficient tool for encoding the levels of a categorical features into numeric values. `LabelEncoder` encode labels with value between 0 and n_classes-1.

Lets encode all the categorical features.

Copy Code

```
# Importing LabelEncoder and initializing it
>> from sklearn.preprocessing import LabelEncoder
>> le=LabelEncoder()
# Iterating over all the common columns in train and test
>> for col in X_test.columns.values:
       # Encoding only categorical variables
       if X_test[col].dtypes=='object':
       # Using whole data to form an exhaustive list of levels
       data=X_train[col].append(X_test[col])
       le.fit(data.values)
       X_train[col]=le.transform(X_train[col])
       X_test[col]=le.transform(X_test[col])
```

All our categorical features are encoded. You can look at your updated data set using `X_train.head()`. We are going to take a look at `Gender` frequency distribution before and after the encoding.

```
Before : Male 318
         Female 66
Name: Gender, dtype: int64

After : 1 318
        0 66
Name: Gender, dtype: int64
```

Now that we are done with label encoding, lets now run a logistic regression model on the data set with both categorical and continuous features.

Copy Code

```
# Standardizing the features
>> X_train_scale=scale(X_train)
>> X_test_scale=scale(X_test)
# Fitting the logistic regression model
>> log=LogisticRegression(penalty='l2',C=.01)
>> log.fit(X_train_scale,Y_train)
# Checking the models accuracy
>> accuracy_score(Y_test,log.predict(X_test_scale))
```

```
Out : 0.75
```

Its working now. But, the accuracy is still the same as we got with logistic regression after standardization from numeric features. This means categorical features we added are not very significant in our objective function.

Exercise 3

Try out decision tree classifier with all the features as independent variables and comment your accuracy.

Resources : Go through this article on decision trees for better understanding.

One-Hot Encoding

One-Hot Encoding transforms each categorical feature with n possible values into n binary features, with only one active.

Most of the ML algorithms either learn a single weight for each feature or it computes distance between the samples. Algorithms like linear models (such as logistic regression) belongs to the first category.

Lets take a look at an example from loan_prediction data set. Feature `Dependents` have 4 possible values 0,1,2 and 3+ which are then encoded without loss of generality to 0,1,2 and 3.

We, then have a weight “W” assigned for this feature in a linear classifier,which will make a decision based on the constraints `W*Dependents + K > 0` or eqivalently `W*Dependents < K`.

Let `f(w)= W*Dependents`

Possible values that can be attained by the equation are 0, W, 2W and 3W. A problem with this equation is that the weight “W” cannot make decision based on four choices. It can reach to a decision in following ways:

- All leads to the same decision (all of them <K or vice versa)
- 3:1 division of the levels (Decision boundary at f(w)>2W)
- 2:2 division of the levels (Decision boundary at f(w)>W)

Here we can see that we are loosing many different possible decisions such as the case where “0” and “2W” should be given same label and “3W” and “W” are odd one out.

This problem can be solved by One-Hot-Encoding as it effectively changes the dimensionality of the feature “Dependents” from one to four, thus every value in the feature “Dependents” will have their own weights. Updated equation for the decison would be `f'(w) < K`.

where, `f'(w) = W1*D_0 + W2*D_1 + W3*D_2 + W4*D_3`

All four new variable has boolean values (0 or 1).

The same thing happens with distance based methods such as kNN. Without encoding, distance between "0" and "1" values of `Dependents` is 1 whereas distance between "0" and "3+" will be 3, which is not desirable as both the distances should be similar. After encoding, the values will be new features (sequence of columns is 0,1,2,3+) : [1,0,0,0] and [0,0,0,1] (initially we were finding distance between "0" and "3+"), now the distance would be √2.

For tree based methods, same situation (more than two values in a feature) might effect the outcome to extent but if methods like random forests are deep enough, it can handle the categorical variables without one-hot encoding.

Now, lets take look at the implementation of one-hot encoding with various algorithms.

Lets create a logistic regression model for classification without one-hot encoding.

Copy Code

```
# We are using scaled variable as we saw in previous section that
# scaling will effect the algo with l1 or l2 reguralizer
>> X_train_scale=scale(X_train)
>> X_test_scale=scale(X_test)
# Fitting a logistic regression model
>> log=LogisticRegression(penalty='l2',C=1)
>> log.fit(X_train_scale,Y_train)
# Checking the model's accuracy
>> accuracy_score(Y_test,log.predict(X_test_scale))
```

```
Out : 0.73958333333333337
```

Now we are going to encode the data.

Copy Code

```
>> from sklearn.preprocessing import OneHotEncoder
>> enc=OneHotEncoder(sparse=False)
>> X_train_1=X_train
>> X_test_1=X_test
>> columns=['Gender', 'Married', 'Dependents', 'Education','Self_Employed',
          'Credit_History', 'Property_Area']
>> for col in columns:
       # creating an exhaustive list of all possible categorical values
```

```
        data=X_train[[col]].append(X_test[[col]])
        enc.fit(data)
        # Fitting One Hot Encoding on train data
        temp = enc.transform(X_train[[col]])
        # Changing the encoded features into a data frame with new column names
        temp=pd.DataFrame(temp,columns=[(col+"_"+str(i)) for i in data[col]
             .value_counts().index])
        # In side by side concatenation index values should be same
        # Setting the index values similar to the X_train data frame
        temp=temp.set_index(X_train.index.values)
        # adding the new One Hot Encoded varibales to the train data frame
        X_train_1=pd.concat([X_train_1,temp],axis=1)
        # fitting One Hot Encoding on test data
        temp = enc.transform(X_test[[col]])
        # changing it into data frame and adding column names
        temp=pd.DataFrame(temp,columns=[(col+"_"+str(i)) for i in data[col]
             .value_counts().index])
        # Setting the index for proper concatenation
        temp=temp.set_index(X_test.index.values)
        # adding the new One Hot Encoded varibales to test data frame
        X_test_1=pd.concat([X_test_1,temp],axis=1)
```

Now, lets apply logistic regression model on one-hot encoded data.

Copy Code

```
# Standardizing the data set
>> X_train_scale=scale(X_train_1)
>> X_test_scale=scale(X_test_1)
# Fitting a logistic regression model
>> log=LogisticRegression(penalty='l2',C=1)
>> log.fit(X_train_scale,Y_train)
# Checking the model's accuracy
>> accuracy_score(Y_test,log.predict(X_test_scale))
```

```
Out : 0.75
```

Here, again we got the maximum accuracy as 0.75 that we have gotten so far. In this case, logistic regression regularization(C) parameter 1 where as earlier we used C=0.01.

End Notes

The aim of this article is to familiarize you with the basic data pre-processing techniques and have a deeper understanding of the situations of where to apply those techniques.

These methods work because of the underlying assumptions of the algorithms. This is by no means an exhaustive list of the methods. I'd encourage you to experiment with these methods since they can be heavily modified according to the problem at hand.

For a more comprehensive guide on data preprocessing, check out our course, "How to Preprocess Data."

I plan to provide more advance techniques of data pre-processing such as pipeline and noise reduction in my next post, so stay tuned to dive deeper into the data pre-processing.

Did you like reading this article ? Do you follow a different approach / package / library to perform these talks. I'd love to interact with you in comments.

You can test your skills and knowledge. Check out Live Competitions and compete with best Data Scientists from all over the world.

S syed

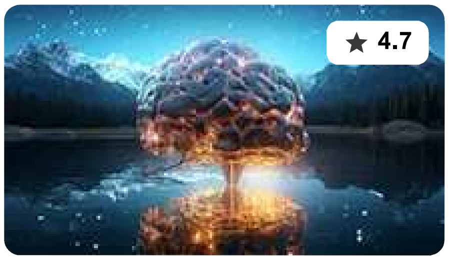
I am Syed Danish, currently pursuing my bachelors in Electronics & Communication Engineering from ISM Dhanbad. I am a data science and machine learning enthusiast.

Business Analytics Classification Data Exploration Intermediate

Libraries Machine Learning Programming Python Python

Structured Data Supervised

Free Courses


★ 4.7

Generative AI - A Way of Life

Explore Generative AI for beginners: create text and images, use top AI tools, learn practical skills, and ethics.

★ 4.5

Getting Started with Large Language Models

Master Large Language Models (LLMs) with this course, offering clear guidance in NLP and model training made simple.

★ 4.6

Responses From Readers

What are your thoughts?...

Submit reply

DR Venugopala Rao Manneni

Useful...

Oluwadara

Thanks for the post. It was useful to me!

Mukul

Feature scaling code X_train[X_train.dtypes[(X_train.dtypes=="float64")|(X_train.dtypes=="int64")].index.values].hist(figsize=[11,11]) is not working. There is something wrong there.

Become an Author

Share insights, grow your voice, and inspire the data community.

- Reach a Global Audience
- Share Your Expertise with the World
- Build Your Brand & Audience

- Join a Thriving AI Community
- Level Up Your AI Game
- Expand Your Influence in Genrative AI

RECOMMENDED ARTICLES

Top 10 Machine Learning Algorithms in 2025

A Comprehensive Guide to Ensemble Learning (wit...

Cook the data for your Machine Learning Algorithm

Organised Preprocessing for Pandas Dataframe

12 Useful Pandas Techniques in Python for Data ...

Practicing Machine Learning Techniques in R wit...

Complete guide on How to learn Scikit-Learn for...

Understand the Concept of Standardization in Ma...

Logistic Regression in Python: Beginner's...

Guide for Building an End-to-End Logistic Regre...

Generative AI Tools and Techniques

GANs | VAEs | Transformers | StyleGAN | Pix2Pix | Autoencoders | GPT | BERT | Word2Vec | LSTM | Attention Mechanisms | Diffusion Models | LLMs | SLMs | Encoder Decoder Models | Prompt Engineering | LangChain | LlamaIndex | RAG | Fine-tuning | LangChain AI Agent | Multimodal Models | RNNs | DCGAN | ProGAN | Text-to-Image Models | DDPM | Document Question Answering | Imagen | T5 (Text-to-Text Transfer Transformer) | Seq2seq Models | WaveNet | Attention Is All You Need (Transformer Architecture) | WindSurf | Cursor

Popular GenAI Models

Llama 4 | Llama 3.1 | GPT 4.5 | GPT 4.1 | GPT 4o | o3-mini | Sora | DeepSeek R1 | DeepSeek V3 | Janus Pro | Veo 2 | Gemini 2.5 Pro | Gemini 2.0 | Gemma 3 | Claude Sonnet 3.7 | Claude 3.5 Sonnet | Phi 4 | Phi 3.5 | Mistral Small 3.1 | Mistral NeMo | Mistral-7b | Bedrock | Vertex AI | Qwen QwQ 32B | Qwen 2 | Qwen 2.5 VL | Qwen Chat | Grok 3

AI Development Frameworks

n8n | LangChain | Agent SDK | A2A by Google | SmolAgents | LangGraph | CrewAI | Agno | LangFlow | AutoGen | LlamaIndex | Swarm | AutoGPT

Data Science Tools and Techniques

Python | R | SQL | Jupyter Notebooks | TensorFlow | Scikit-learn | PyTorch | Tableau | Apache Spark | Matplotlib | Seaborn | Pandas | Hadoop | Docker | Git | Keras | Apache Kafka | AWS | NLP | Random Forest | Computer Vision | Data Visualization | Data Exploration | Big Data | Common Machine Learning Algorithms | Machine Learning | Google Data Science Agent

Company

Discover