Application Building
Import required libraries

Date	20 October 2022
Team ID	PNT2022TMID434346
Project Name	Project - Web Phishing Detection

Import required libraries

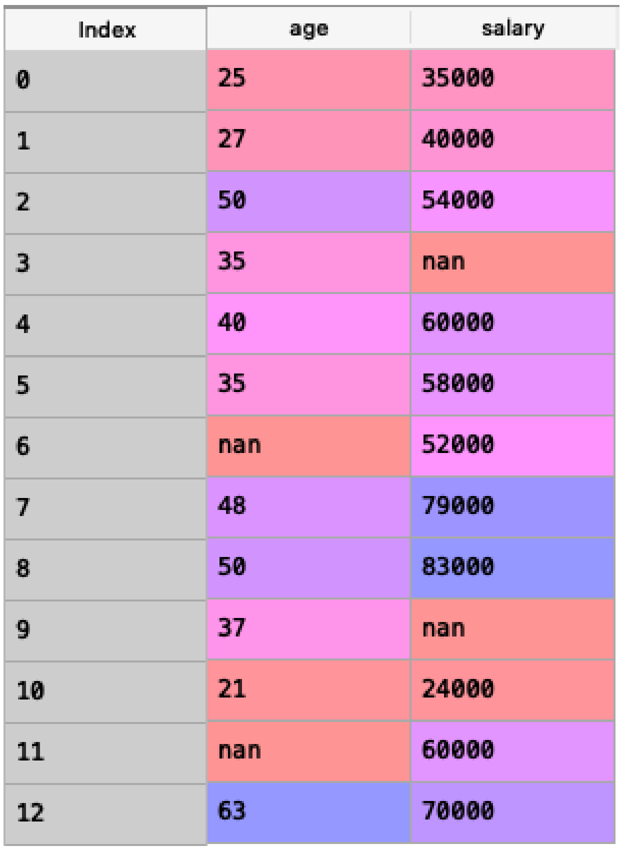
Importing the pandas

```
import pandas as pd
```

Loading the dataset

```
dataset = pd.read_excel("age_salary.xls")
```

The data set used here is as simple as shown below:



Index	age	salary
0	25	35000
1	27	40000
2	50	54000
3	35	nan
4	40	60000
5	35	58000
6	nan	52000
7	48	79000
8	50	83000
9	37	nan
10	21	24000
11	nan	60000
12	63	70000

Note:

The ‘nan’ you see in some cells of the dataframe denotes the missing fields

Now that we have loaded our dataset lets play with it.

Classifying the dependent and Independent Variables

Having seen the data we can clearly identify the dependent and independent factors.Here we just have 2 factors, age and salary.Salary is the dependent factor that changes with the independent factor age.Now let’s classify them programmatically.

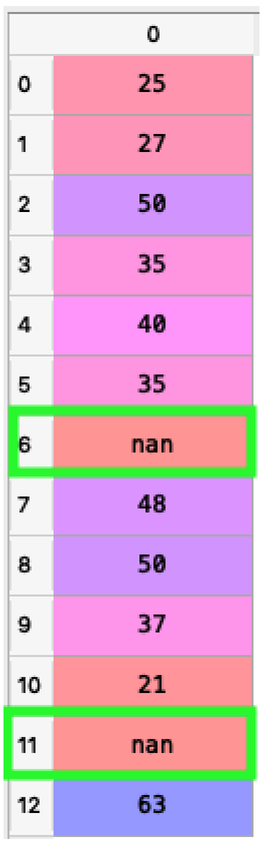
```
X = dataset.iloc[:,:-1].values #Takes all rows of all columns except the last column
Y = dataset.iloc[:,-1].values # Takes all rows of the last column
```

- X : independent variable set
- Y : dependent variable set

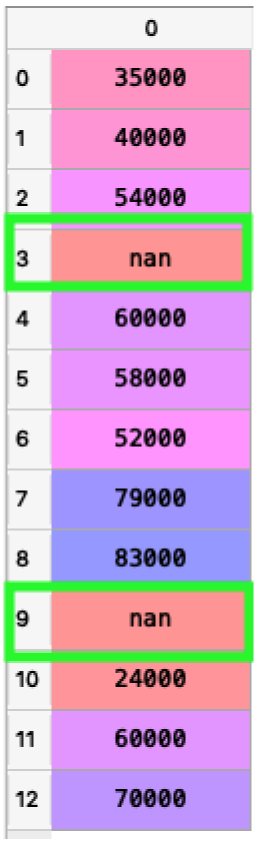
The dependent and independent values are stored in different arrays. In case of multiple independent variables use

```
X = dataset.iloc[:,a:b].values
```

where a is the starting range and b is the ending range (column indices). You can also specify the column indices in a list to select specific columns.



	0
0	25
1	27
2	50
3	35
4	40
5	35
6	nan
7	48
8	50
9	37
10	21
11	nan
12	63



	0
0	35000
1	40000
2	54000
3	nan
4	60000
5	58000
6	52000
7	79000
8	83000
9	nan
10	24000
11	60000
12	70000

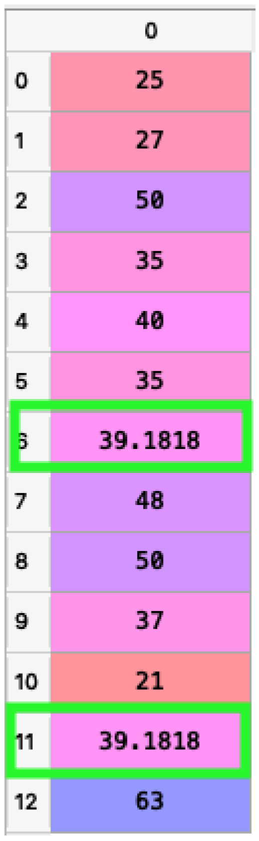
Dealing with Missing Data

We have already noticed the missing fields in the data denoted by "nan". Machine learning models cannot accommodate missing fields in the data they are provided with.So the missing fields must be filled with values that will not affect the variance of the data or make it more noisy.

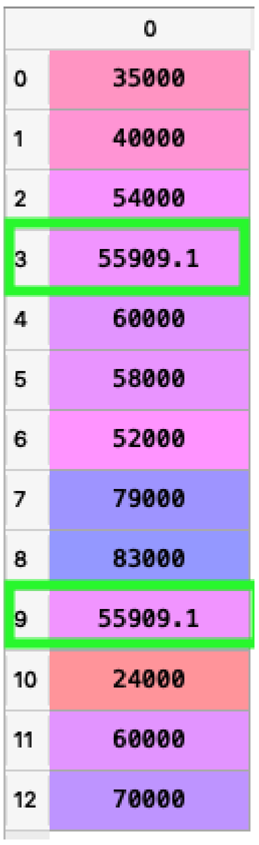
```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan,
strategy="mean")
X = imp.fit_transform(X)
Y = Y.reshape(-1,1)
Y = imp.fit_transform(Y)
Y = Y.reshape(-1)
```

The scikit-learn library's SimpleImputer Class allows us to impute the missing fields in a dataset with valid data. In the above code, we have used the default strategy for filling missing values which is the mean. The imputer can not be applied on 1D arrays and since Y is a 1D array, it needs to be converted to a compatible shape.The reshape functions allows us to reshape any array.The fit_transform() method will fit the imputer object and then transforms the arrays.

Output



	0
0	25
1	27
2	50
3	35
4	40
5	35
6	39.1818
7	48
8	50
9	37
10	21
11	39.1818
12	63

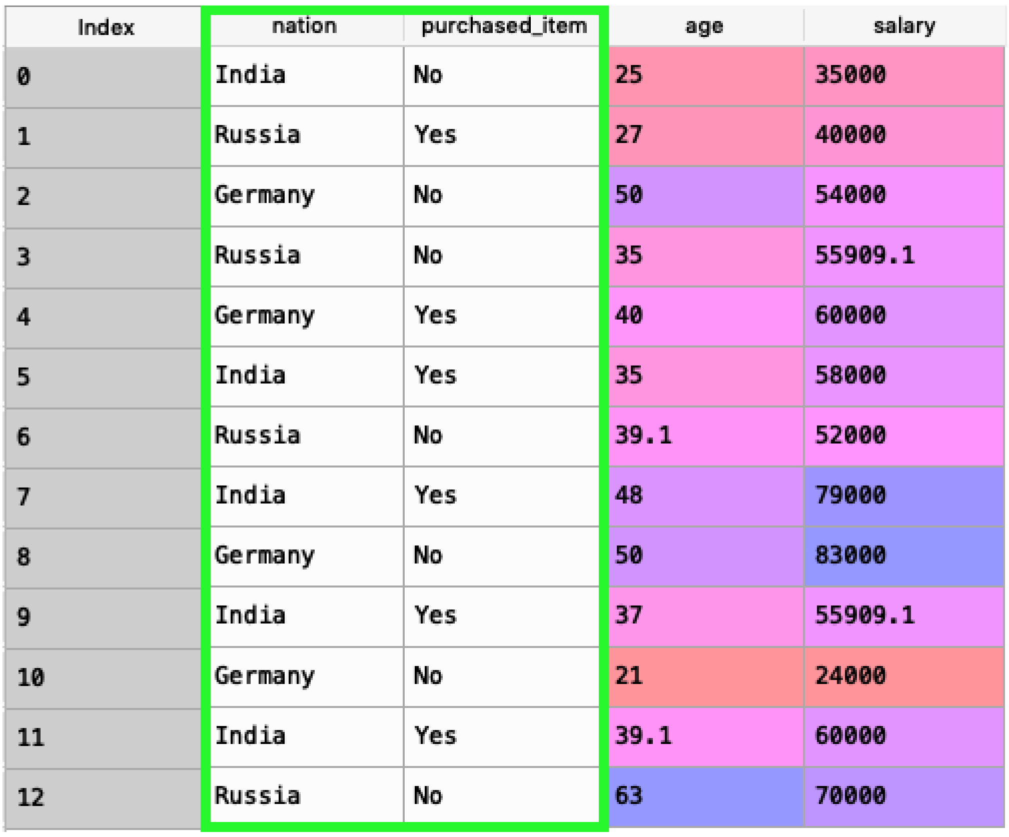


	0
0	35000
1	40000
2	54000
3	55909.1
4	60000
5	58000
6	52000
7	79000
8	83000
9	55909.1
10	24000
11	60000
12	70000

Dealing with Categorical Data

When dealing with large and real-world datasets, categorical data is almost inevitable.Categorical variables represent types of data which may be divided into groups. Examples of categorical variables are race, sex, age group, educational level etc. These variables often has letters or words as its values. Since machine learning models are all about numbers and calculations , these categorical variables need to be coded in to numbers. Having coded the categorical variable into numbers may just not be enough.

For example, consider the dataset below with 2 categorical features nation and purchased_item. Let us assume that the dataset is a record of how age, salary and country of a person determine if an item is purchased or not.Thus purchased_item is the dependent factor and age, salary and nation are the independent factors.

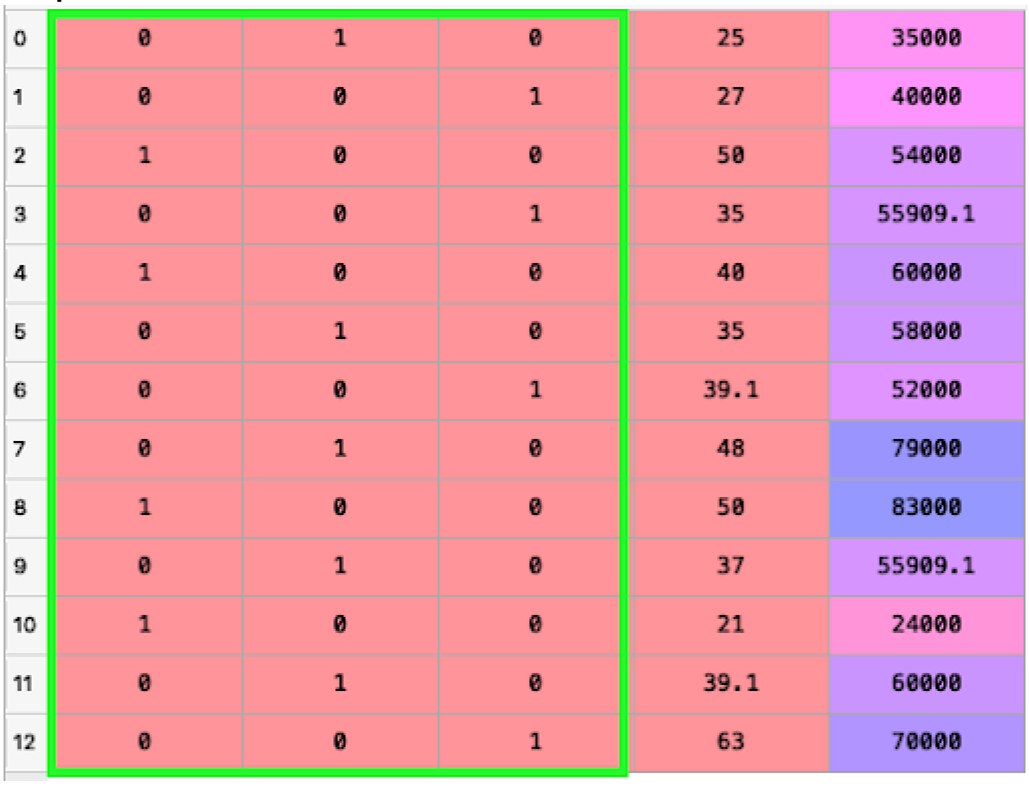


Index	nation	purchased_item	age	salary
0	India	No	25	35000
1	Russia	Yes	27	40000
2	Germany	No	50	54000
3	Russia	No	35	55909.1
4	Germany	Yes	40	60000
5	India	Yes	35	58000
6	Russia	No	39.1	52000
7	India	Yes	48	79000
8	Germany	No	50	83000
9	India	Yes	37	55909.1
10	Germany	No	21	24000
11	India	Yes	39.1	60000
12	Russia	No	63	70000

It has 3 countries listed. In a larger dataset, these may be large groups of data. Since countries don't have a mathematical relation between them unless we are considering some known factors such as size or population etc , coding them in numbers will not work, as a number may be less than or greater than another number. Dummy variables are the solution. Using one hot encoding we will create a dummy variable for each of the category in the column. And uses binary encoding for each dummy variable. We do not need to create dummy variables for the feature purchased_item as it has only 2 categories either yes or no.

```
dataset = pd.read_csv("dataset.csv")
X = dataset.iloc[:,[0,2,3]].values
Y = dataset.iloc[:,1].values
from sklearn.preprocessing import
LabelEncoder,OneHotEncoder
le_X = LabelEncoder()
X[:,0] = le_X.fit_transform(X[:,0])
ohe_X = OneHotEncoder(categorical_features =
[0])
X = ohe_X.fit_transform(X).toarray()
```

Output

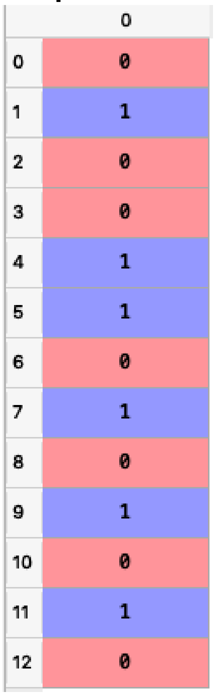


0	0	1	0	25	35000
1	0	0	1	27	40000
2	1	0	0	50	54000
3	0	0	1	35	55909.1
4	1	0	0	40	60000
5	0	1	0	35	58000
6	0	0	1	39.1	52000
7	0	1	0	48	79000
8	1	0	0	50	83000
9	0	1	0	37	55909.1
10	1	0	0	21	24000
11	0	1	0	39.1	60000
12	0	0	1	63	70000

The the first 3 columns are the dummy features representing Germany,India and Russia respectively.The 1’s in each column represent that the person belongs to that specific country.

```
Y = le_X.fit_transform(Y)
```

Output:



	0
0	0
1	1
2	0
3	0
4	1
5	1
6	0
7	1
8	0
9	1
10	0
11	1
12	0

Splitting the Dataset into Training and Testing sets

All machine learning models require us to provide a training set for the machine so that the model can train from that data to understand the relations between features and can predict for new observations.When we are provided a single huge dataset with too much of observations ,it is a good idea to split the dataset into to two, a training_set and a test_set, so that we can test our model after its been trained with the training_set.

Scikit-learn comes with a method called train_test_split to help us with this task.

```
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)
```

The above code will split X and Y into two subsets each.

- test_size: the desired size of the test_set. 0.3 denotes 30%.
- random_state: This is used to preserve the uniqueness. The split will happen uniquely for a random_state.

Scaling the features

Since machine learning models rely on numbers to solve relations it is important to have similarly scaled data in a dataset. Scaling ensures that all data in a dataset falls in the same range.Unscaled data can cause inaccurate or false predictions.Some machine learning algorithms can handle feature scaling on its own and doesn’t require it explicitly.

The StandardScaler class from the scikit-learn library can help us scale the dataset.

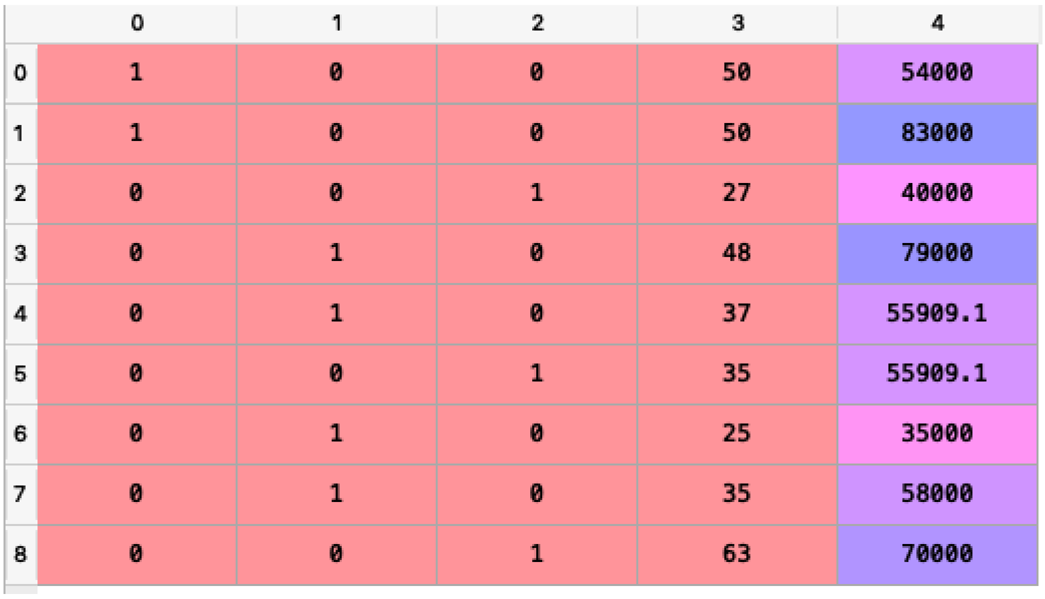
```
from sklearn.preprocessing import
StandardScaler
sc_X = StandardScaler()
```

```
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

sc_y = StandardScaler()
Y_train = Y_train.reshape((len(Y_train), 1))
Y_train = sc_y.fit_transform(Y_train)
Y_train = Y_train.ravel()
```

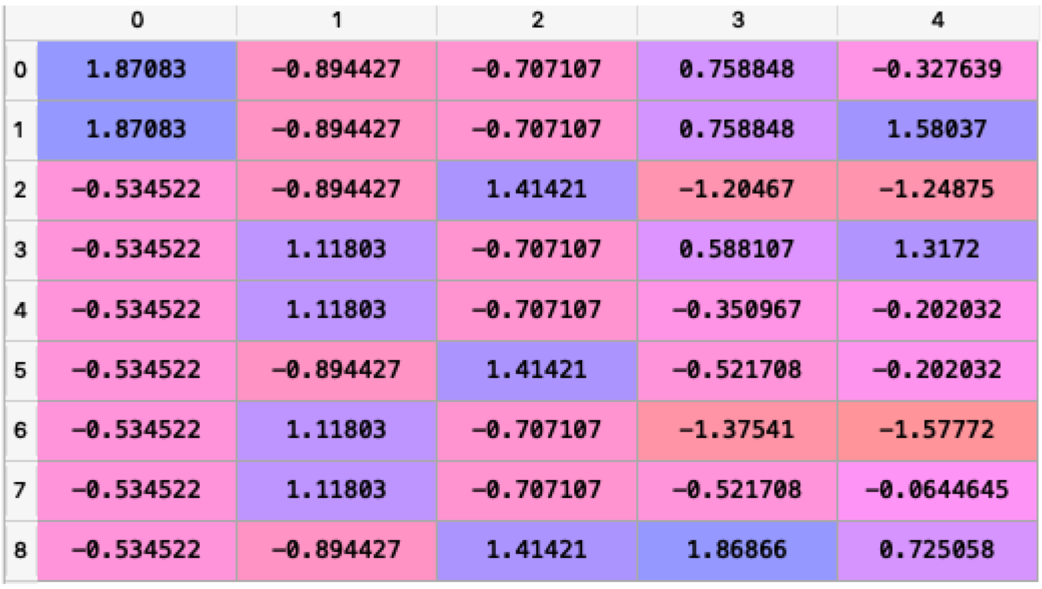
Output

X_train before scaling :



	0	1	2	3	4
0	1	0	0	50	54000
1	1	0	0	50	83000
2	0	0	1	27	40000
3	0	1	0	48	79000
4	0	1	0	37	55909.1
5	0	0	1	35	55909.1
6	0	1	0	25	35000
7	0	1	0	35	58000
8	0	0	1	63	70000

X_train after scaling :



	0	1	2	3	4
0	1.87083	-0.894427	-0.707107	0.758848	-0.327639
1	1.87083	-0.894427	-0.707107	0.758848	1.58037
2	-0.534522	-0.894427	1.41421	-1.20467	-1.24875
3	-0.534522	1.11803	-0.707107	0.588107	1.3172
4	-0.534522	1.11803	-0.707107	-0.350967	-0.202032
5	-0.534522	-0.894427	1.41421	-0.521708	-0.202032
6	-0.534522	1.11803	-0.707107	-1.37541	-1.57772
7	-0.534522	1.11803	-0.707107	-0.521708	-0.0644645
8	-0.534522	-0.894427	1.41421	1.86866	0.725058