Category Machine learning projects

Dec 7, 2019 (http://www.datasciencelovers.com/2019/12/)

Logistic regression to predict absenteeism- approach (http://www.datasciencelovers.com/machine-learning-projects/logistic-regression-to-predict-absenteeism-approach/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/logistic-regression-to-predict-absenteeism-approach/)**
Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/) Tag **absenteeism prediction (http://www.datasciencelovers.com/tag/absenteeism-prediction/)**, **classification (http://www.datasciencelovers.com/tag/classification/)**, **logistic regression (http://www.datasciencelovers.com/tag/logistic-regression/)**, **machine learning (http://www.datasciencelovers.com/tag/machine-learning/)**, **python-sql-Tableau (http://www.datasciencelovers.com/tag/python-sql-tableau/)**

Business Problem:

In today environment there is a high competitiveness which increase pressure on employee. High competitiveness leads unachievable goals, which cause an employee health issues, and health issue will lead absenteeism of employee.

With a given dataset an organisation is trying to predict employee absenteeism.

What is absenteeism in the business context?

Absence from work during normal working hours, resulting in temporary incapacity to execute regular working activity.

Purpose Of Model:

Explore whether a person presenting certain characteristics is expected to be away from work at some points in time or not.

Dataset:

I have downloaded a data set from kaggle called '**Absenteeism_data.csv'** which contain following information.

- **Reason_1** – A Type of Reason to be absent.
- **Reason_2** – A Type of Reason to be absent.
- **Reason_3** – A Type of Reason to be absent.
- **Reason_4** – A Type of Reason to be absent.
- **Month Value** – Month in which employee has been absent.
- **Day of the Week** – Days
- **Transportation Expense** – Expense in dollar
- **Distance to Work** – Distance of workplace in Km
- **Age** – Age of employee
- **Daily Work Load Average** – Average amount of time spent working per day shown in minutes.
- **Body Mass Index** – Body Mass index of employee.
- **Education** – Education category(1 – high school education, 2 – Graduate, 3 – Post graduate, 4 – A Master or Doctor)
- **Children** – No of children an employee has
- **Pet** – Whether employee has pet or not?
- **Absenteeism Time in Hours** – How many hours an employee has been absent.

Following are the main action we will take in this project.

1. **Build the model in python**
2. **Save the result in Mysql.**
3. **Visualise the end result in Tableau**

Python for model building:

We are going to take following steps to predict absenteeism:

Load the data

Import the '**Absenteeism_data.csv'** with the help of pandas

Identify dependent Variable i.e. identify the Y:

We have to be categories and we must find a way to say if someone is 'being absent too much' or not. what we've decided to do is to take the median of the dataset as a cut-off line in this way the dataset will be balanced (there will be roughly equal number of 0s and 1s for the logistic regression) as balancing is a great problem for ML, this will work great for us alternatively, if we had more data, we could have found other ways to deal with the issue for instance, we could have assigned some arbitrary value as a cut-off line, instead of the median.

Note that what line does is to assign 1 to anyone who has been absent 4 hours or more (more than 3 hours) that is the equivalent of taking half a day off initial code from the lecture targets = np.where(data_preprocessed['Absenteeism Time in Hours'] > 3, 1, 0)

Choose Algorithm to develop model:

As our Y (dependent variable) is 1 or o i.e. absent or not absent so we are going to use Logistic regression for our analysis.

Select Input for the regression:

We have to select our all x variables i.e. all independent variable which we will use for regression analysis.

Data Pre-processing:

Remove or treat missing value

In our case there is no missing value so we don't have to worry about missing value. Yes, there are some columns who is not adding any value in our analysis such as ID which is unique in every case so we will remove it.

Remove Outliers

In our case there are no outliers so we don't have to worry. But in general if you have outlier you can take log of your x variable to remove outliers.

Standardize the data

standardization is one of the most common pre-processing tools since data of different magnitude (scale) can be biased towards high values, we want all inputs to be of similar magnitude this is a peculiarity of machine learning in general – most (but not all) algorithms do badly with unscaled data. A very useful module we can use is Standard Scaler. It has much more capabilities than the straightforward 'pre-processing' method. We will create a variable that will contain the scaling information for this particular dataset.

Here's the full documentation:

http://scikitlearn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (http://scikitlearn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

Choose the column to scales

In this section we need to choose that variable which need to transform or scale.in our case we need to scale ['Reason_1', 'Reason_2', 'Reason_3', 'Reason_4','Education', pet and 'children'], because these are the columns which contain categorical data but in numerical form so we need to transform them.

What about the other column?

'Month Value', 'Day of the Week', 'Transportation Expense', 'Distance to Work', 'Age', 'Daily Work Load Average', 'Body Mass Index' . These are the numerical value and their data type is int. so we do not have to transform them but will keep in our analysis.

Note:-

You can ask why we are doing analysis manually column wise?

Because it is always good to analyse data feature wise it gives us a confidence for our model and we can easily interpret our model analysis.

Split Data into train and test

Divide our data into train and test and build the model on train data set.

Apply Algorithm

As per our scenario we are going to use logistic regression in our case. Following steps will take place

Train the model

First we will divide the data into train and test. We will build our model on train data set.

Test the model

When we successfully developed our model then we need to test with a new data set which is testing data sets.

Find the intercepts and coefficient

Find out the beta values and coefficient from model.

Interpreting the coefficients

Find out which feature is adding more values in predictions of Y.

Save the model

Need to save the model which we have prepared so far. To do that we need to pickle the model.

Two executable file will save in your python directory one '**model'** and the other is '**scaler'**

To save your .Ipnyb file in form of executable, save the same as .py file.

Check Model performance on totally new data set with same features.

Now we have a totally new data set which has same feature as per previous data set but contain different values.

Note – To do that your executable file '**model'**, **scaler' and '.py' file should be in same folder.**

Mysql for Data store

Save the prediction in data base (Mysql)

It is always good to save data and prediction on centralised data base. So create a data base in mysql and create a table with all field available in your predicted data frame i.e '**df_new_obs'**

Import '**pymysql'** library to make connection between ipynb notebook and mysql.

Setup the connection with user name and password and insert the predicted output values. In the data base.

Tableau for Data visualization

Connect the data base with Tableau and visualize the result

As we know tableau is a strong tool to visualise the data. So in our case we will connect our database with tableau and visualise our result and present to the business.

To connect tableau with my sql we need to take following steps.

- Open the tableau desktop application.
- Click on connect data source as mysql.
- Put your data base address, username and password.
- Select the data base.
- Drag the table and visualize your data.

Dec 7, 2019 (http://www.datasciencelovers.com/2019/12/)

Absenteeism prediction implementation with Python Part-1 (http://www.datasciencelovers.com/machine-learning-projects/absenteeism-prediction-implementation-with-python/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/absenteeism-prediction-implementation-with-python/)**
Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/) Tag **absenteeism prediction (http://www.datasciencelovers.com/tag/absenteeism-prediction/)**, **classification (http://www.datasciencelovers.com/tag/classification/)**, **logistic regression (http://www.datasciencelovers.com/tag/logistic-regression/)**, **python-sql-Tableau (http://www.datasciencelovers.com/tag/python-sql-tableau/)**

Now we will see here how to develop algorithm to predict absenteeism in python.

So, lets see the jupyter notebook file how I have done it.

Creating a logistic regression to predict absenteeism

Import the relevant libraries

In [1]:
```
# import the relevant libraries
import pandas as pd
import numpy as np
```

Load the data

In [5]:
```
# load the preprocessed CSV data
data_preprocessed = pd.read_csv('Absenteeism_preprocessed.csv')
```

In [6]:
```
# eyeball the data
data_preprocessed.head()
```

Out[6]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month Value	Day of the Week	Transportation Expense	Distan to Wo
0	0	0	0	1	7	1	289	
1	0	0	0	0	7	1	118	
2	0	0	0	1	7	2	179	
3	1	0	0	0	7	3	279	
4	0	0	0	1	7	3	289	

In [7]:
```
data_preprocessed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 15 columns):
Reason_1                    700 non-null int64
Reason_2                    700 non-null int64
Reason_3                    700 non-null int64
Reason_4                    700 non-null int64
Month Value                 700 non-null int64
Day of the Week             700 non-null int64
Transportation Expense      700 non-null int64
Distance to Work            700 non-null int64
Age                         700 non-null int64
Daily Work Load Average     700 non-null float64
Body Mass Index             700 non-null int64
Education                   700 non-null int64
Children                    700 non-null int64
Pet                         700 non-null int64
```

```
Absenteeism Time in Hours    700 non-null int64
dtypes: float64(1), int64(14)
memory usage: 82.1 KB
```

Create the targets

In [8]:
```
# find the median of 'Absenteeism Time in Hours'
data_preprocessed['Absenteeism Time in Hours'].median()
```

Out[8]:
```
3.0
```

In [9]:
```
# what we've decided to do is to take the median of the dataset as
a cut-off line
# in this way the dataset will be balanced (there will be roughly e
qual number of 0s and 1s for the logistic regression)

# note that what line does is to assign 1 to anyone who has been ab
sent 4 hours or more (more than 3 hours)
# that is the equivalent of taking half a day off

# targets = np.where(data_preprocessed['Absenteeism Time in Hours']
> 3, 1, 0)

# parameterized code
targets = np.where(data_preprocessed['Absenteeism Time in Hours'] >
                   data_preprocessed['Absenteeism Time in Hours'].m
edian(), 1, 0)
```

In [10]:
```
# eyeball the targets
targets
```

Out[10]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
1, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1,
1, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1,
       0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0,
0, 1,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1,
0, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,
1, 1,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
0, 0,
       0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
0, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
```

```
0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0,
0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
0, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
0, 1,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
0, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1,
0, 1,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0,
0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,
1, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0])
```

In [11]:
```
# create a Series in the original data frame that will contain the
targets for the regression
data_preprocessed['Excessive Absenteeism'] = targets
```

In [12]:
```
# check what happened
# maybe manually see how the targets were created
data_preprocessed.head()
```

Out[12]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month Value	Day of the Week	Transportation Expense	Distan to Wo
0	0	0	0	1	7	1	289	
1	0	0	0	0	7	1	118	
2	0	0	0	1	7	2	179	
3	1	0	0	0	7	3	279	
4	0	0	0	1	7	3	289	

A comment on the targets

In [13]:
```
# check if dataset is balanced (what % of targets are 1s)
# targets.sum() will give us the number of 1s that there are
# the shape[0] will give us the length of the targets array
targets.sum() / targets.shape[0]
```

Out[13]: 0.45571428571428574

In [14]:
```
# create a checkpoint by dropping the unnecessary variables
# also drop the variables we 'eliminated' after exploring the weigh
ts
data_with_targets = data_preprocessed.drop(['Absenteeism Time in Ho
urs','Day of the Week',
                                            'Daily Work Load Averag
e','Distance to Work'],axis=1)
```

In [15]:
```
# check if the line above is a checkpoint :)

# if data_with_targets is data_preprocessed = True, then the two ar
e pointing to the same object
# if it is False, then the two variables are completely different a
nd this is in fact a checkpoint
data_with_targets is data_preprocessed
```

Out[15]: False

In [16]:
```
# check what's inside
data_with_targets.head()
```

Out[16]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month Value	Transportation Expense	Age	Body Mass Index
0	0	0	0	1	7	289	33	30
1	0	0	0	0	7	118	50	31
2	0	0	0	1	7	179	38	31
3	1	0	0	0	7	279	39	24
4	0	0	0	1	7	289	33	30

Select the inputs for the regression

In [17]:
```
data_with_targets.shape
```

Out[17]: (700, 12)

In [18]:
```
# Selects all rows and all columns until 14 (excluding)
data_with_targets.iloc[:,:14]
```

Out[18]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month Value	Transportation Expense	Age	Body Mass Index

0	0	0	0	1	7	289	33	30
1	0	0	0	0	7	118	50	31
2	0	0	0	1	7	179	38	31
3	1	0	0	0	7	279	39	24
4	0	0	0	1	7	289	33	30
5	0	0	0	1	10	179	38	31
6	0	0	0	1	7	361	28	27
7	0	0	0	1	7	260	36	23
8	0	0	1	0	6	155	34	25
9	0	0	0	1	7	235	37	29
10	1	0	0	0	7	260	36	23
11	1	0	0	0	7	260	36	23
12	1	0	0	0	7	260	36	23
13	1	0	0	0	7	179	38	31
14	0	0	0	1	7	179	38	31
15	1	0	0	0	7	246	41	23
16	0	0	0	1	7	179	38	31
17	0	0	1	0	7	179	38	31
18	1	0	0	0	7	189	33	25
19	0	0	0	1	5	248	47	32
20	1	0	0	0	12	330	28	25
21	1	0	0	0	3	179	38	31
22	1	0	0	0	10	361	28	27
23	0	0	0	1	8	260	36	23
24	0	0	1	0	8	289	33	30
25	0	0	0	1	8	361	28	27
26	0	0	0	1	4	289	33	30
27	0	0	0	1	12	157	29	22
28	0	0	1	0	8	289	33	30
29	0	0	0	1	8	179	38	31
...	...	...	...	...	...	...	...	...
670	0	0	0	1	4	155	34	25
671	0	0	1	0	4	225	28	24
672	1	0	0	0	4	118	50	31
673	0	0	0	1	4	179	30	19
674	0	0	0	1	7	235	37	29
675	0	0	1	0	9	225	41	28
676	0	0	0	1	9	235	32	25
677	1	0	0	0	9	118	37	28
678	0	0	0	1	9	235	43	38

679	1	0	0	0	10	179	30	19
680	0	0	0	1	10	291	40	25
681	1	0	0	0	10	225	41	28
682	0	0	1	0	11	300	43	25
683	0	0	0	1	11	225	41	28
684	0	0	0	1	11	179	30	19
685	0	0	0	1	5	118	50	31
686	1	0	0	0	5	118	50	31
687	0	0	0	1	5	118	37	28
688	0	0	0	0	5	118	50	31
689	0	0	0	1	5	179	30	19
690	0	0	0	0	5	378	36	21
691	0	1	0	0	5	179	40	22
692	1	0	0	0	5	155	34	25
693	1	0	0	0	5	235	32	25
694	0	0	0	1	5	291	40	25
695	1	0	0	0	5	179	40	22
696	1	0	0	0	5	225	28	24
697	1	0	0	0	5	330	28	25
698	0	0	0	1	5	235	32	25
699	0	0	0	1	5	291	40	25

700 rows × 12 columns

In [19]:
```
# Selects all rows and all columns but the last one (basically the same operation)
data_with_targets.iloc[:,:-1]
```

Out[19]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month Value	Transportation Expense	Age	Body Mass Index
0	0	0	0	1	7	289	33	30
1	0	0	0	0	7	118	50	31
2	0	0	0	1	7	179	38	31
3	1	0	0	0	7	279	39	24
4	0	0	0	1	7	289	33	30
5	0	0	0	1	10	179	38	31
6	0	0	0	1	7	361	28	27
7	0	0	0	1	7	260	36	23
8	0	0	1	0	6	155	34	25
9	0	0	0	1	7	235	37	29
10	1	0	0	0	7	260	36	23

11	1	0	0	0	7	260	36	23
12	1	0	0	0	7	260	36	23
13	1	0	0	0	7	179	38	31
14	0	0	0	1	7	179	38	31
15	1	0	0	0	7	246	41	23
16	0	0	0	1	7	179	38	31
17	0	0	1	0	7	179	38	31
18	1	0	0	0	7	189	33	25
19	0	0	0	1	5	248	47	32
20	1	0	0	0	12	330	28	25
21	1	0	0	0	3	179	38	31
22	1	0	0	0	10	361	28	27
23	0	0	0	1	8	260	36	23
24	0	0	1	0	8	289	33	30
25	0	0	0	1	8	361	28	27
26	0	0	0	1	4	289	33	30
27	0	0	0	1	12	157	29	22
28	0	0	1	0	8	289	33	30
29	0	0	0	1	8	179	38	31
...	...	...	...	...	...	...	...	...
670	0	0	0	1	4	155	34	25
671	0	0	1	0	4	225	28	24
672	1	0	0	0	4	118	50	31
673	0	0	0	1	4	179	30	19
674	0	0	0	1	7	235	37	29
675	0	0	1	0	9	225	41	28
676	0	0	0	1	9	235	32	25
677	1	0	0	0	9	118	37	28
678	0	0	0	1	9	235	43	38
679	1	0	0	0	10	179	30	19
680	0	0	0	1	10	291	40	25
681	1	0	0	0	10	225	41	28
682	0	0	1	0	11	300	43	25
683	0	0	0	1	11	225	41	28
684	0	0	0	1	11	179	30	19
685	0	0	0	1	5	118	50	31
686	1	0	0	0	5	118	50	31
687	0	0	0	1	5	118	37	28
688	0	0	0	0	5	118	50	31
689	0	0	0	1	5	179	30	19

690	0	0	0	0	5	378	36	21
691	0	1	0	0	5	179	40	22
692	1	0	0	0	5	155	34	25
693	1	0	0	0	5	235	32	25
694	0	0	0	1	5	291	40	25
695	1	0	0	0	5	179	40	22
696	1	0	0	0	5	225	28	24
697	1	0	0	0	5	330	28	25
698	0	0	0	1	5	235	32	25
699	0	0	0	1	5	291	40	25

700 rows × 11 columns

In [20]:
```
# Create a variable that will contain the inputs (everything withou
t the targets)
unscaled_inputs = data_with_targets.iloc[:,:-1]
```

Standardize the data

In [21]:
```
#Import library
from sklearn.preprocessing import StandardScaler


# define scaler as an object
absenteeism_scaler = StandardScaler()
```

In [22]:
```
# import the libraries needed to create the Custom Scaler
# note that all of them are a part of the sklearn package

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler

# create the Custom Scaler class

class CustomScaler(BaseEstimator,TransformerMixin):

    # init or what information we need to declare a CustomScaler ob
ject
    # and what is calculated/declared as we do

    def __init__(self,columns,copy=True,with_mean=True,with_std=Tru
e):

        # scaler is nothing but a Standard Scaler object
        self.scaler = StandardScaler(copy,with_mean,with_std)
        # with some columns 'twist'
        self.columns = columns
        self.mean_ = None
        self.var_ = None


    # the fit method, which, again based on StandardScale
```

```
    # the fit method, which, again based on StandardScaler

    def fit(self, X, y=None):
        self.scaler.fit(X[self.columns], y)
        self.mean_ = np.mean(X[self.columns])
        self.var_ = np.var(X[self.columns])
        return self

    # the transform method which does the actual scaling

    def transform(self, X, y=None, copy=None):

        # record the initial order of the columns
        init_col_order = X.columns

        # scale all features that you chose when creating the instance of the class
        X_scaled = pd.DataFrame(self.scaler.transform(X[self.columns]), columns=self.columns)

        # declare a variable containing all information that was not scaled
        X_not_scaled = X.loc[:,~X.columns.isin(self.columns)]

        # return a data frame which contains all scaled features and all 'not scaled' features
        # use the original order (that you recorded in the beginning)
        return pd.concat([X_not_scaled, X_scaled], axis=1)[init_col_order]
```

In [23]:
```
# check what are all columns that we've got
unscaled_inputs.columns.values
```

Out[23]:
```
array(['Reason_1', 'Reason_2', 'Reason_3', 'Reason_4', 'Month Value',
       'Transportation Expense', 'Age', 'Body Mass Index', 'Education',
       'Children', 'Pet'], dtype=object)
```

In [63]:
```
# choose the columns to scale
# we later augmented this code and put it in comments
# columns_to_scale = ['Month Value','Day of the Week', 'Transportation Expense', 'Distance to Work',
       #'Age', 'Daily Work Load Average', 'Body Mass Index', 'Children', 'Pet']

# select the columns to omit
columns_to_omit = ['Reason_1', 'Reason_2', 'Reason_3', 'Reason_4','Education']
```

In [26]:
```
# create the columns to scale, based on the columns to omit
# use list comprehension to iterate over the list
columns_to_scale = [x for x in unscaled_inputs.columns.values if x not in columns_to_omit]
```

In [27]:
```
# declare a scaler object, specifying the columns you want to scale
absenteeism_scaler = CustomScaler(columns_to_scale)
```

In [28]:
```
# fit the data (calculate mean and standard deviation); they are au
```

```
tomatically stored inside the object
absenteeism_scaler.fit(unscaled_inputs)
```

Out[28]:
```
CustomScaler(columns=['Month Value', 'Transportation Expense', 'Ag
e', 'Body Mass Index', 'Children', 'Pet'],
       copy=None, with_mean=None, with_std=None)
```

In [29]:
```
# standardizes the data, using the transform method
# in the last line, we fitted the data - in other words
# we found the internal parameters of a model that will be used to
transform data.
# transforming applies these parameters to our data
# note that when we get new data, we can just call 'scaler' again a
nd transform it in the same way as now
scaled_inputs = absenteeism_scaler.transform(unscaled_inputs)
```

In [30]:
```
# the scaled_inputs are now an ndarray, because sklearn works with
ndarrays
scaled_inputs
```

Out[30]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month Value	Transportation Expense	Ag
0	0	0	0	1	0.030796	1.005844	-0.53606
1	0	0	0	0	0.030796	-1.574681	2.13080
2	0	0	0	1	0.030796	-0.654143	0.24831
3	1	0	0	0	0.030796	0.854936	0.40518
4	0	0	0	1	0.030796	1.005844	-0.53606
5	0	0	0	1	0.929019	-0.654143	0.24831
6	0	0	0	1	0.030796	2.092381	-1.32043
7	0	0	0	1	0.030796	0.568211	-0.06543
8	0	0	1	0	-0.268611	-1.016322	-0.37918
9	0	0	0	1	0.030796	0.190942	0.09143
10	1	0	0	0	0.030796	0.568211	-0.06543
11	1	0	0	0	0.030796	0.568211	-0.06543
12	1	0	0	0	0.030796	0.568211	-0.06543
13	1	0	0	0	0.030796	-0.654143	0.24831
14	0	0	0	1	0.030796	-0.654143	0.24831
15	1	0	0	0	0.030796	0.356940	0.71893
16	0	0	0	1	0.030796	-0.654143	0.24831
17	0	0	1	0	0.030796	-0.654143	0.24831
18	1	0	0	0	0.030796	-0.503235	-0.53606
19	0	0	0	1	-0.568019	0.387122	1.66018
20	1	0	0	0	1.527833	1.624567	-1.32043
21	1	0	0	0	-1.166834	-0.654143	0.24831
22	1	0	0	0	0.929019	2.092381	-1.32043
23	0	0	0	1	0.330204	0.568211	-0.06543

24	0	0	1	0	0.330204	1.005844	-0.53606
25	0	0	0	1	0.330204	2.092381	-1.32043
26	0	0	0	1	-0.867426	1.005844	-0.53606
27	0	0	0	1	1.527833	-0.986140	-1.16356
28	0	0	1	0	0.330204	1.005844	-0.53606
29	0	0	0	1	0.330204	-0.654143	0.24831
...	...	...	...	...	...	...	.
670	0	0	0	1	-0.867426	-1.016322	-0.37918
671	0	0	1	0	-0.867426	0.040034	-1.32043
672	1	0	0	0	-0.867426	-1.574681	2.13080
673	0	0	0	1	-0.867426	-0.654143	-1.00668
674	0	0	0	1	0.030796	0.190942	0.09143
675	0	0	1	0	0.629611	0.040034	0.71893
676	0	0	0	1	0.629611	0.190942	-0.69293
677	1	0	0	0	0.629611	-1.574681	0.09143
678	0	0	0	1	0.629611	0.190942	1.03268
679	1	0	0	0	0.929019	-0.654143	-1.00668
680	0	0	0	1	0.929019	1.036026	0.56205
681	1	0	0	0	0.929019	0.040034	0.71893
682	0	0	1	0	1.228426	1.171843	1.03268
683	0	0	0	1	1.228426	0.040034	0.71893
684	0	0	0	1	1.228426	-0.654143	-1.00668
685	0	0	0	1	-0.568019	-1.574681	2.13080
686	1	0	0	0	-0.568019	-1.574681	2.13080
687	0	0	0	1	-0.568019	-1.574681	0.09143
688	0	0	0	0	-0.568019	-1.574681	2.13080
689	0	0	0	1	-0.568019	-0.654143	-1.00668
690	0	0	0	0	-0.568019	2.348925	-0.06543
691	0	1	0	0	-0.568019	-0.654143	0.56205
692	1	0	0	0	-0.568019	-1.016322	-0.37918
693	1	0	0	0	-0.568019	0.190942	-0.69293
694	0	0	0	1	-0.568019	1.036026	0.56205
695	1	0	0	0	-0.568019	-0.654143	0.56205
696	1	0	0	0	-0.568019	0.040034	-1.32043
697	1	0	0	0	-0.568019	1.624567	-1.32043
698	0	0	0	1	-0.568019	0.190942	-0.69293
699	0	0	0	1	-0.568019	1.036026	0.56205

700 rows × 11 columns

In [31]:
```
# check the shape of the inputs
```

```
scaled_inputs.shape
```

Out[31]: (700, 11)

Split the data into train & test and shuffle

Import the relevant module

In [32]:
```
# import train_test_split so we can split our data into train and t
est
from sklearn.model_selection import train_test_split
```

Split

In [33]:
```
# check how this method works
train_test_split(scaled_inputs, targets)
```

Out[33]:
```
[     Reason_1  Reason_2  Reason_3  Reason_4  Month Value  \
 407         0         0         0         0    -1.166834
 473         0         0         0         1     0.030796
 582         1         0         0         0    -1.765648
 498         0         0         0         1    -0.568019
 694         0         0         0         1    -0.568019
 63          0         0         0         1     0.929019
 43          0         0         1         0     0.330204
 286         0         0         0         1     0.629611
 540         0         0         0         1     1.228426
 160         1         0         0         0    -1.466241
 83          0         0         1         0     1.527833
 148         0         0         0         1    -1.466241
 691         0         1         0         0    -0.568019
 342         0         0         0         1    -0.568019
 249         0         0         1         0     1.228426
 576         1         0         0         0     1.228426
 300         0         0         0         0     0.929019
 668         0         1         0         0    -0.867426
 346         0         0         0         1     1.527833
 393         0         0         0         1     0.929019
 660         1         0         0         0     0.629611
 333         0         0         0         1     1.228426
 688         0         0         0         0    -0.568019
 692         1         0         0         0    -0.568019
 572         0         0         0         1    -0.867426
 54          0         0         0         0     0.629611
 296         1         0         0         0     1.527833
 20          1         0         0         0     1.527833
 295         1         0         0         0     1.527833
 287         1         0         0         0     0.629611
 ..        ...       ...       ...       ...          ...
 199         1         0         0         0    -0.867426
 284         0         0         0         1     0.629611
 345         1         0         0         0     1.527833
 504         0         0         0         1     0.629611
 227         1         0         0         0    -0.268611
```

```
 418         0         0         0         1     1.527833
 138         1         0         0         0    -1.166834
 200         0         0         1         0    -0.867426
 30          0         0         1         0     0.330204
 336         0         0         0         0     1.228426
 608         0         0         0         1    -1.466241
 596         1         0         0         0    -1.466241
 689         0         0         0         1    -0.568019
 143         0         0         0         1     1.527833
 23          0         0         0         1     0.330204
 191         1         0         0         0    -0.268611
 559         0         0         0         1     0.330204
 308         0         0         0         1     0.929019
 39          0         0         0         1     0.330204
 205         0         0         0         1    -0.568019
 6           0         0         0         1     0.030796
 164         1         0         0         0     0.030796
 626         0         0         0         1     0.330204
 621         0         0         0         1    -0.268611
 508         1         0         0         0    -0.568019
 283         0         0         0         1     0.629611
 555         1         0         0         0    -0.568019
 338         0         0         0         1     1.228426
 149         0         0         0         1    -1.466241
 304         0         0         0         1     0.929019

      Transportation Expense       Age  Body Mass Index  Education
Children  \
 407               -1.574681  2.130803         1.002633          0
-0.019280
 473               -1.574681  2.130803         1.002633          0
-0.019280
 582                1.005844 -0.536062         0.767431          0
0.880469
 498               -1.016322 -0.379188        -0.408580          0
0.880469
 694                1.036026  0.562059        -0.408580          0
-0.019280
 63                -1.574681  0.091435         0.297027          0
-0.919030
 43                 0.190942  1.032682         2.649049          0
-0.019280
 286                1.036026  0.562059        -0.408580          0
-0.019280
 540                2.092381 -1.320435         0.061825          0
-0.019280
 160                0.568211 -0.065439        -0.878984          0
2.679969
 83                -0.654143  0.562059        -1.114186          1
0.880469
 148               -0.654143 -1.006686        -1.819793          1
-0.919030
 691               -0.654143  0.562059        -1.114186          1
0.880469
 342                0.190942  0.091435         0.532229          1
-0.019280
 249                1.005844 -0.536062         0.767431          0
0.880469
 576                1.005844  1.973929         2.178644          0
-0.919030
 300                0.190942  1.032682         2.649049          0
-0.019280
```

```
 668                0.190942 -0.692937        -0.408580          1
-0.919030
 346               -0.654143  0.248310         1.002633          0
-0.919030
 393                0.568211 -0.065439        -0.878984          0
2.679969
 660                2.213108 -0.849811        -0.408580          0
1.780219
 333               -0.654143  0.248310         1.002633          0
-0.919030
 688               -1.574681  2.130803         1.002633          0
-0.019280
 692               -1.016322 -0.379188        -0.408580          0
0.880469
 572               -0.654143  0.562059        -1.114186          1
0.880469
 54                 1.005844 -0.536062         0.767431          0
0.880469
 296               -1.574681  0.091435         0.297027          0
-0.919030
 20                 1.624567 -1.320435        -0.408580          1
-0.919030
 295               -0.654143 -1.006686        -1.819793          1
-0.919030
 287                1.036026  0.562059        -0.408580          0
-0.019280
 ..                      ...       ...              ...        ...
...
 199               -1.016322 -0.379188        -0.408580          0
0.880469
 284               -1.574681  2.130803         1.002633          0
-0.019280
 345                0.356940  0.718933        -0.878984          0
-0.919030
 504               -0.654143 -1.006686        -1.819793          1
-0.919030
 227                0.356940  0.718933        -0.878984          0
-0.919030
 418               -0.654143  0.248310         1.002633          0
-0.919030
 138                0.356940  0.718933        -0.878984          0
-0.919030
 200                2.348925 -0.065439        -1.349389          0
0.880469
 30                -0.654143  0.248310         1.002633          0
-0.919030
 336                2.348925 -0.065439        -1.349389          0
0.880469
 608               -1.016322 -0.379188        -0.408580          0
0.880469
 596                0.040034 -1.320435        -0.643782          0
-0.019280
 689               -0.654143 -1.006686        -1.819793          1
-0.919030
 143                1.005844 -0.536062         0.767431          0
0.880469
 23                 0.568211 -0.065439        -0.878984          0
2.679969
 191               -0.654143  0.248310         1.002633          0
-0.919030
 559                0.040034 -1.320435        -0.643782          0
-0.019280
```

```
 0.019280
 308               -0.654143 -1.006686        -1.819793          1
-0.919030
 39                 0.568211 -0.065439        -0.878984          0
2.679969
 205               -1.016322 -0.379188        -0.408580          0
0.880469
 6                  2.092381 -1.320435         0.061825          0
-0.019280
 164               -0.654143  0.562059        -1.114186          1
0.880469
 626                0.040034 -1.320435        -0.643782          0
-0.019280
 621                0.387122  1.660180         1.237836          0
0.880469
 508                0.190942 -0.692937        -0.408580          1
-0.919030
 283                0.190942  1.032682         2.649049          0
-0.019280
 555               -0.654143  0.248310         1.002633          0
-0.919030
 338               -0.654143  0.248310         1.002633          0
-0.919030
 149               -0.578689 -1.477309        -1.349389          0
-0.919030
 304                0.190942  1.032682         2.649049          0
-0.019280

          Pet
 407 -0.589690
 473 -0.589690
 582  0.268487
 498 -0.589690
 694  0.268487
 63  -0.589690
 43  -0.589690
 286  0.268487
 540  2.843016
 160 -0.589690
 83  -0.589690
 148 -0.589690
 691 -0.589690
 342  0.268487
 249  0.268487
 576  1.126663
 300 -0.589690
 668 -0.589690
 346 -0.589690
 393 -0.589690
 660 -0.589690
 333 -0.589690
 688 -0.589690
 692 -0.589690
 572 -0.589690
 54   0.268487
 296 -0.589690
 20  -0.589690
 295 -0.589690
 287  0.268487
 ..        ...
 199 -0.589690
 284 -0.589690
```

```
345 -0.589690
504 -0.589690
227 -0.589690
418 -0.589690
138 -0.589690
200  2.843016
30  -0.589690
336  2.843016
608 -0.589690
596  1.126663
689 -0.589690
143  0.268487
23  -0.589690
191 -0.589690
559  1.126663
308 -0.589690
39  -0.589690
205 -0.589690
6    2.843016
164 -0.589690
626  1.126663
621  0.268487
508 -0.589690
283 -0.589690
555 -0.589690
338 -0.589690
149 -0.589690
304 -0.589690

[525 rows x 11 columns],
     Reason_1  Reason_2  Reason_3  Reason_4  Month Value  \
604         0         0         0         1    -1.466241
134         0         0         0         1    -1.765648
563         0         0         0         1     1.527833
615         0         0         0         1    -1.466241
224         1         0         0         0    -0.268611
231         1         0         0         0     0.330204
614         0         0         0         1    -1.466241
360         0         0         0         1     0.629611
386         0         0         0         1    -1.466241
232         0         0         0         1     0.629611
126         0         0         0         1    -1.765648
221         0         0         0         1    -0.268611
419         0         0         0         1    -0.867426
642         0         0         0         1    -1.166834
322         1         0         0         0     1.228426
190         0         0         0         1    -0.867426
455         1         0         0         0    -0.268611
509         0         0         0         1    -0.268611
658         1         0         0         0    -0.568019
632         0         0         0         1    -1.166834
254         0         0         0         1     0.030796
233         1         0         0         0     0.929019
76          0         0         0         1     0.929019
382         0         0         0         1    -1.466241
33          0         0         1         0     0.330204
123         0         0         0         1     1.527833
178         1         0         0         0    -1.166834
538         1         0         0         0     1.228426
500         0         0         0         1     0.330204
121         0         0         0         1     1.228426
..        ...       ...       ...       ...          ...
```

400	0	0	0	0	-1.166834
678	0	0	0	1	0.629611
430	0	0	0	1	1.228426
341	0	0	0	1	-0.568019
193	0	0	0	1	0.330204
404	1	0	0	0	-1.166834
397	0	0	0	1	-1.166834
277	0	0	0	0	0.629611
591	0	0	0	1	0.330204
476	0	0	0	1	0.030796
166	0	0	0	1	-1.166834
480	0	0	0	1	-1.166834
607	0	0	0	1	-1.466241
319	1	0	0	0	0.330204
465	0	0	0	1	0.030796
128	0	0	0	1	1.527833
489	0	0	0	1	0.330204
428	0	0	0	1	0.929019
466	0	0	0	1	0.030796
49	1	0	0	0	0.629611
657	0	0	1	0	-0.867426
309	0	0	0	1	0.929019
198	0	0	1	0	-0.867426
332	1	0	0	0	1.228426
181	0	0	0	1	-1.166834
469	0	0	0	1	0.030796
429	0	0	0	1	0.929019
443	0	0	0	1	-0.268611
352	1	0	0	0	1.527833
630	1	0	0	0	-1.166834

	Transportation Expense	Age	Body Mass Index	Education	Children
604	-0.654143	-1.006686	-1.819793	1	-0.919030
134	-1.574681	0.091435	0.297027	0	-0.919030
563	0.190942	1.032682	2.649049	0	-0.019280
615	-0.654143	0.248310	1.002633	0	-0.919030
224	0.356940	0.718933	-0.878984	0	-0.919030
231	-1.574681	2.130803	1.002633	0	-0.019280
614	0.040034	-1.320435	-0.643782	0	-0.019280
360	-1.016322	-0.379188	-0.408580	0	0.880469
386	-0.654143	0.248310	1.002633	0	-0.919030
232	2.348925	-0.065439	-1.349389	0	0.880469
126	-1.574681	0.091435	0.297027	0	-0.919030
221	2.092381	-1.320435	0.061825	0	-0.019280
419	-1.574681	2.130803	1.002633	0	-0.019280
642	0.387122	1.660180	1.237836	0	0.880469
322	2.348925	-0.065439	-1.349389	0	

```
 322                2.348925 -0.065439        -1.349389          0
0.880469
 190                0.568211 -0.065439        -0.878984          0
2.679969
 455               -0.654143  0.248310         1.002633          0
-0.919030
 509                0.356940  0.718933        -0.878984          0
-0.919030
 658               -0.503235 -0.536062        -0.408580          0
0.880469
 632               -0.654143  0.248310         1.002633          0
-0.919030
 254                1.005844 -0.536062         0.767431          0
0.880469
 233               -0.654143  0.248310         1.002633          0
-0.919030
 76                 0.040034 -1.320435        -0.643782          0
-0.019280
 382                0.356940  0.718933        -0.878984          0
-0.919030
 33                 0.190942  1.817054         1.473038          0
-0.019280
 123               -1.574681  0.091435         0.297027          0
-0.919030
 178                0.040034 -1.320435        -0.643782          0
-0.019280
 538                0.190942 -0.692937        -0.408580          1
-0.919030
 500               -0.654143 -1.006686        -1.819793          1
-0.919030
 121               -1.574681  0.091435         0.297027          0
-0.919030
 ..                      ...       ...              ...        ...
...
 400                2.213108 -0.849811        -0.408580          0
1.780219
 678                0.190942  1.032682         2.649049          0
-0.019280
 430                2.092381 -1.320435         0.061825          0
-0.019280
 341               -0.654143  0.248310         1.002633          0
-0.919030
 193                1.036026  0.562059        -0.408580          0
-0.019280
 404               -1.574681  2.130803         1.002633          0
-0.019280
 397                2.348925 -0.065439        -1.349389          0
0.880469
 277                1.036026 -0.692937        -0.878984          0
-0.919030
 591               -0.654143 -1.006686        -1.819793          1
-0.919030
 476                0.190942  1.032682         2.649049          0
-0.019280
 166                0.568211 -0.065439        -0.878984          0
2.679969
 480               -0.654143  0.248310         1.002633          0
-0.919030
 607               -0.654143  0.248310         1.002633          0
-0.919030
 319               -0.654143  0.248310         1.002633          0
-0.919030
```

```
 465               -1.574681  0.091435         0.297027          0
-0.919030
 128               -1.574681  0.091435         0.297027          0
-0.919030
 489                1.624567 -1.320435        -0.408580          1
-0.919030
 428               -0.654143 -1.006686        -1.819793          1
-0.919030
 466                0.568211 -0.065439        -0.878984          0
2.679969
 49                 1.036026  0.562059        -0.408580          0
-0.019280
 657                0.387122  1.660180         1.237836          0
0.880469
 309               -1.574681  2.130803         1.002633          0
-0.019280
 198                1.005844 -0.536062         0.767431          0
0.880469
 332                0.356940  0.718933        -0.878984          0
-0.919030
 181                0.040034 -1.320435        -0.643782          0
-0.019280
 469                0.190942  1.817054         1.473038          0
-0.019280
 429               -0.654143  0.248310         1.002633          0
-0.919030
 443               -1.574681  0.091435         0.297027          0
-0.919030
 352                0.190942  1.032682         2.649049          0
-0.019280
 630               -0.654143  0.248310         1.002633          0
-0.919030

          Pet
 604 -0.589690
 134 -0.589690
 563 -0.589690
 615 -0.589690
 224 -0.589690
 231 -0.589690
 614  1.126663
 360 -0.589690
 386 -0.589690
 232  2.843016
 126 -0.589690
 221  2.843016
 419 -0.589690
 642  0.268487
 322  2.843016
 190 -0.589690
 455 -0.589690
 509 -0.589690
 658  1.126663
 632 -0.589690
 254  0.268487
 233 -0.589690
 76   1.126663
 382 -0.589690
 33   3.701192
 123 -0.589690
 178  1.126663
 538 -0.589690
```

```
500 -0.589690
121 -0.589690
..        ...
400 -0.589690
678 -0.589690
430  2.843016
341 -0.589690
193  0.268487
404 -0.589690
397  2.843016
277 -0.589690
591 -0.589690
476 -0.589690
166 -0.589690
480 -0.589690
607 -0.589690
319 -0.589690
465 -0.589690
128 -0.589690
489 -0.589690
428 -0.589690
466 -0.589690
49   0.268487
657  0.268487
309 -0.589690
198  0.268487
332 -0.589690
181  1.126663
469  3.701192
429 -0.589690
443 -0.589690
352 -0.589690
630 -0.589690

[175 rows x 11 columns],
array([0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1,
1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0,
1, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
0, 0,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
0, 0,
       0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
0, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
0, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
1, 1,
       0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
1, 0,
       0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1,
1, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1
```

```
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1,
0, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
1, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1,
0, 0,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
0, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0]),
 array([0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0,
0, 0,
       0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1,
0, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
0, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1,
1, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0,
1, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1,
0, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1,
1])]
```

In [34]:
```
# declare 4 variables for the split
x_train, x_test, y_train, y_test = train_test_split(scaled_inputs,
targets, #train_size = 0.8,

test_size = 0.2, random_state = 20)
```

In [35]:
```
# check the shape of the train inputs and targets
print (x_train.shape, y_train.shape)
```

```
(560, 11) (560,)
```

In [36]:
```
# check the shape of the test inputs and targets
print (x_test.shape, y_test.shape)
```

```
(140, 11) (140,)
```

Logistic regression with sklearn

In [37]:
```
# import the LogReg model from sklearn
from sklearn.linear_model import LogisticRegression
```

```
# import the 'metrics' module, which includes important metrics we
may want to use
from sklearn import metrics
```

Training the model

In [38]:
```
# create a logistic regression object
reg = LogisticRegression()
```

In [39]:
```
# fit our train inputs
# that is basically the whole training part of the machine learning
reg.fit(x_train,y_train)
```

Out[39]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interc
ept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_j
obs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=
0.0001,
          verbose=0, warm_start=False)
```

In [40]:
```
# assess the train accuracy of the model
reg.score(x_train,y_train)
```

Out[40]:
```
0.775
```

Manually check the accuracy

In [41]:
```
# find the model outputs according to our model
model_outputs = reg.predict(x_train)
model_outputs
```

Out[41]:
```
array([0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0,
1, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1,
1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
1, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
1, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0,
0, 1,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
0, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
0, 1,
       0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
```

```
1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1,
0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
1, 1,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
0, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1,
0, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0,
0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1,
1, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1,
1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
0, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1,
0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0,
1, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0])
```

In [42]:
```
# compare them with the targets
y_train
```

Out[42]:
```
array([0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
0, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
1, 1,
       1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,
0, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
1, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
0, 1,
       0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
1, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
0, 1,
       1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1,
0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0,
0, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1,
0, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
1, 1,
```

```
       1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0,
0, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
0, 0,
       1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0,
1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
1, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
1, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1,
0, 0,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
1, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
0, 0,
       0, 0, 0, 1, 1, 1, 1, 0, 1, 0])
```

In [43]:
```
# ACTUALLY compare the two variables
model_outputs == y_train
```

Out[43]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  Tru
e,
        True, False,  True, False, False,  True,  True,  True,  Tru
e,
       False,  True, False,  True, False, False,  True,  True,  Tru
e,
       False,  True,  True,  True,  True,  True,  True,  True,  Tru
e,
       False, False, False, False,  True,  True,  True,  True, Fals
e,
        True,  True,  True,  True,  True, False,  True,  True,  Tru
e,
        True,  True,  True,  True,  True, False,  True,  True,  Tru
e,
        True,  True,  True,  True,  True,  True, False,  True,  Tru
e,
        True,  True,  True, False,  True,  True,  True,  True,  Tru
e,
       False,  True, False,  True,  True, False, False, False,  Tru
e,
        True,  True,  True,  True,  True,  True,  True, False,  Tru
e,
        True,  True,  True,  True,  True,  True,  True,  True,  Tru
e,
        True,  True,  True,  True, False,  True,  True,  True,  Tru
e,
       False,  True,  True,  True,  True, False,  True,  True,  Tru
e,
        True,  True,  True,  True,  True, False,  True,  True,  Tru
e,
        True, False,  True,  True,  True,  True,  True,  True, Fals
e,
        True, False,  True, False,  True,  True,  True,  True, Fals
e,
       False, False,  True,  True, False,  True,  True,  True,  Tru
e,
```

```
         True, False,  True, False,  True, False,  True, False, Fals
e,
         True,  True,  True,  True,  True,  True,  True,  True,  Tru
e,
         True,  True,  True,  True,  True, False,  True,  True,  Tru
e,
         True, False,  True,  True,  True,  True,  True,  True,  Tru
e,
         True,  True,  True,  True,  True,  True, False,  True, Fals
e,
        False,  True,  True,  True,  True,  True,  True,  True, Fals
e,
         True, False,  True, False,  True,  True,  True,  True, Fals
e,
         True, False, False,  True,  True,  True,  True,  True, Fals
e,
        False, False,  True, False,  True,  True,  True, False,  Tru
e,
         True,  True,  True,  True,  True,  True, False,  True,  Tru
e,
         True,  True,  True,  True,  True,  True, False,  True,  Tru
e,
         True, False, False,  True,  True,  True,  True,  True,  Tru
e,
        False,  True,  True,  True,  True,  True,  True, False, Fals
e,
        False,  True,  True,  True,  True, False,  True, False,  Tru
e,
        False,  True,  True,  True,  True,  True, False,  True, Fals
e,
        False,  True,  True,  True,  True,  True, False,  True,  Tru
e,
         True,  True,  True, False,  True, False,  True,  True,  Tru
e,
         True,  True,  True,  True,  True, False,  True,  True, Fals
e,
        False,  True,  True,  True,  True,  True,  True,  True, Fals
e,
         True,  True,  True, False, False,  True,  True,  True, Fals
e,
         True, False,  True,  True,  True,  True,  True,  True,  Tru
e,
         True,  True,  True,  True,  True,  True,  True,  True,  Tru
e,
        False,  True,  True, False,  True, False,  True,  True,  Tru
e,
         True,  True,  True,  True,  True,  True,  True,  True,  Tru
e,
         True,  True,  True, False,  True,  True, False,  True,  Tru
e,
         True, False,  True, False,  True,  True,  True, False,  Tru
e,
         True,  True,  True,  True,  True, False,  True,  True,  Tru
e,
         True,  True,  True,  True,  True,  True,  True,  True,  Tru
e,
        False,  True,  True, False,  True, False,  True,  True, Fals
e,
         True,  True,  True, False,  True,  True, False,  True, Fals
e,
         True,  True,  True,  True,  True, False, False,  True,  Tru
e,
```

```
        True,  True,  True,  True,  True,  True,  True, False,  Tru
e,
       False,  True,  True,  True, False, False,  True,  True,  Tru
e,
        True, False,  True,  True,  True,  True,  True,  True,  Tru
e,
        True,  True, False,  True,  True, False, False,  True,  Tru
e,
       False,  True,  True,  True,  True,  True,  True, False, Fals
e,
        True,  True, False,  True,  True,  True,  True, False,  Tru
e,
        True,  True,  True,  True, False,  True,  True,  True,  Tru
e,
        True, False,  True,  True, False,  True,  True,  True,  Tru
e,
       False,  True,  True,  True,  True,  True, False,  True,  Tru
e,
       False, False, False,  True,  True, False,  True,  True,  Tru
e,
       False,  True,  True,  True,  True,  True,  True,  True,  Tru
e,
        True, False,  True,  True,  True,  True,  True,  True,  Tru
e,
        True,  True, False,  True,  True,  True,  True, False,  Tru
e,
       False,  True])
```

In [44]:
```
# find out in how many instances we predicted correctly
np.sum((model_outputs==y_train))
```

Out[44]:
```
434
```

In [45]:
```
# get the total number of instances
model_outputs.shape[0]
```

Out[45]:
```
560
```

In [46]:
```
# calculate the accuracy of the model
np.sum((model_outputs==y_train)) / model_outputs.shape[0]
```

Out[46]:
```
0.775
```

Finding the intercept and coefficients

In [47]:
```
# get the intercept (bias) of our model
reg.intercept_
```

Out[47]:
```
array([-1.43138127])
```

In [48]:
```
# get the coefficients (weights) of our model
reg.coef_
```

Out[48]:
```
array([[ 2.60237227,  0.84350002,  2.94078723,  0.63723433,  0.0056
5051,
         0.61953401, -0.17635497,  0.28410321, -0.26372527,  0.3519
5032,
        -0.27369766]])
```

In [49]:
```
# check what were the names of our columns
unscaled_inputs.columns.values
```

Out[49]:
```
array(['Reason_1', 'Reason_2', 'Reason_3', 'Reason_4', 'Month Valu
e',
       'Transportation Expense', 'Age', 'Body Mass Index', 'Educati
on',
       'Children', 'Pet'], dtype=object)
```

In [50]:
```
# save the names of the columns in an ad-hoc variable
feature_name = unscaled_inputs.columns.values
```

In [51]:
```
# use the coefficients from this table (they will be exported later
and will be used in Tableau)
# transpose the model coefficients (model.coef_) and throws them in
to a df (a vertical organization, so that they can be
# multiplied by certain matrices later)
summary_table = pd.DataFrame (columns=['Feature name'], data = feat
ure_name)

# add the coefficient values to the summary table
summary_table['Coefficient'] = np.transpose(reg.coef_)

# display the summary table
summary_table
```

Out[51]:

	Feature name	Coefficient
0	Reason_1	2.602372
1	Reason_2	0.843500
2	Reason_3	2.940787
3	Reason_4	0.637234
4	Month Value	0.005651
5	Transportation Expense	0.619534
6	Age	-0.176355
7	Body Mass Index	0.284103
8	Education	-0.263725
9	Children	0.351950
10	Pet	-0.273698

In [52]:
```
# do a little Python trick to move the intercept to the top of the
summary table
# move all indices by 1
summary_table.index = summary_table.index + 1

# add the intercept at index 0
summary_table.loc[0] = ['Intercept', reg.intercept_[0]]

# sort the df by index
summary_table = summary_table.sort_index()
summary_table
```

Out[52]:

	Feature name	Coefficient

	Feature name	Coefficient
0	Intercept	-1.431381
1	Reason_1	2.602372
2	Reason_2	0.843500
3	Reason_3	2.940787
4	Reason_4	0.637234
5	Month Value	0.005651
6	Transportation Expense	0.619534
7	Age	-0.176355
8	Body Mass Index	0.284103
9	Education	-0.263725
10	Children	0.351950
11	Pet	-0.273698

Interpreting the coefficients

In [53]:
```
# create a new Series called: 'Odds ratio' which will show the.. od
ds ratio of each feature
summary_table['Odds_ratio'] = np.exp(summary_table.Coefficient)
```

In [54]:
```
# display the df
summary_table
```

Out[54]:

	Feature name	Coefficient	Odds_ratio
0	Intercept	-1.431381	0.238979
1	Reason_1	2.602372	13.495716
2	Reason_2	0.843500	2.324489
3	Reason_3	2.940787	18.930743
4	Reason_4	0.637234	1.891243
5	Month Value	0.005651	1.005667
6	Transportation Expense	0.619534	1.858062
7	Age	-0.176355	0.838320
8	Body Mass Index	0.284103	1.328570
9	Education	-0.263725	0.768185
10	Children	0.351950	1.421838
11	Pet	-0.273698	0.760562

In [55]:
```
# sort the table according to odds ratio
# note that by default, the sort_values method sorts values by 'asc
ending'
summary_table.sort_values('Odds_ratio', ascending=False)
```

Out[55]:

	Feature name	Coefficient	Odds_ratio

3	Reason_3	2.940787	18.930743
1	Reason_1	2.602372	13.495716
2	Reason_2	0.843500	2.324489
4	Reason_4	0.637234	1.891243
6	Transportation Expense	0.619534	1.858062
10	Children	0.351950	1.421838
8	Body Mass Index	0.284103	1.328570
5	Month Value	0.005651	1.005667
7	Age	-0.176355	0.838320
9	Education	-0.263725	0.768185
11	Pet	-0.273698	0.760562
0	Intercept	-1.431381	0.238979

Testing the model

In [56]:
```
# assess the test accuracy of the model
reg.score(x_test,y_test)
```

Out[56]:
```
0.7357142857142858
```

In [57]:
```
# find the predicted probabilities of each class
# the first column shows the probability of a particular observatio
n to be 0, while the second one - to be 1
predicted_proba = reg.predict_proba(x_test)

# let's check that out
predicted_proba
```

Out[57]:
```
array([[0.75308922, 0.24691078],
       [0.60926091, 0.39073909],
       [0.4859575 , 0.5140425 ],
       [0.7552847 , 0.2447153 ],
       [0.0839675 , 0.9160325 ],
       [0.30192695, 0.69807305],
       [0.30166774, 0.69833226],
       [0.1151045 , 0.8848955 ],
       [0.73775967, 0.26224033],
       [0.75403176, 0.24596824],
       [0.50719215, 0.49280785],
       [0.19719276, 0.80280724],
       [0.06163196, 0.93836804],
       [0.70917025, 0.29082975],
       [0.29280547, 0.70719453],
       [0.5241047 , 0.4758953 ],
       [0.50676929, 0.49323071],
       [0.50888352, 0.49111648],
       [0.367008  , 0.632992  ],
       [0.06355661, 0.93644339],
       [0.73644831, 0.26355169],
       [0.7552847 , 0.2447153 ],
       [0.47457156, 0.52542844],
       [0.47288443, 0.52711557],
       [0.22026535, 0.77973465],
```

```
[0.73808685, 0.26191315],
[0.51184512, 0.48815488],
[0.87683579, 0.12316421],
[0.23445563, 0.76554437],
[0.7552847 , 0.2447153 ],
[0.61087074, 0.38912926],
[0.28414073, 0.71585927],
[0.29943679, 0.70056321],
[0.50634641, 0.49365359],
[0.7552847 , 0.2447153 ],
[0.40453482, 0.59546518],
[0.73743223, 0.26256777],
[0.21439711, 0.78560289],
[0.56310493, 0.43689507],
[0.39544424, 0.60455576],
[0.75559726, 0.24440274],
[0.50110763, 0.49889237],
[0.73874043, 0.26125957],
[0.55298784, 0.44701216],
[0.19310398, 0.80689602],
[0.39168937, 0.60831063],
[0.27751475, 0.72248525],
[0.75465877, 0.24534123],
[0.75144366, 0.24855634],
[0.75590955, 0.24409045],
[0.49772404, 0.50227596],
[0.67413797, 0.32586203],
[0.30192695, 0.69807305],
[0.75302024, 0.24697976],
[0.17932441, 0.82067559],
[0.60845511, 0.39154489],
[0.09280493, 0.90719507],
[0.73302148, 0.26697852],
[0.64208684, 0.35791316],
[0.64169795, 0.35830205],
[0.29600002, 0.70399998],
[0.3008583 , 0.6991417 ],
[0.73202705, 0.26797295],
[0.21881596, 0.78118404],
[0.75380354, 0.24619646],
[0.75340367, 0.24659633],
[0.90974125, 0.09025875],
[0.73710452, 0.26289548],
[0.2325724 , 0.7674276 ],
[0.70777255, 0.29222745],
[0.73939295, 0.26060705],
[0.64716344, 0.35283656],
[0.11680764, 0.88319236],
[0.56393718, 0.43606282],
[0.406166  , 0.593834  ],
[0.7552847 , 0.2447153 ],
[0.23415211, 0.76584789],
[0.25308247, 0.74691753],
[0.29723426, 0.70276574],
[0.36543731, 0.63456269],
[0.73841377, 0.26158623],
[0.9123835 , 0.0876165 ],
[0.73003114, 0.26996886],
[0.28139582, 0.71860418],
[0.53366197, 0.46633803],
[0.87738286, 0.12261714],
[0.30050256, 0.69949744],
```

```
[0.30090256, 0.69949744],
[0.47288443, 0.52711557],
[0.7464465 , 0.2535535 ],
[0.28414073, 0.71585927],
[0.80468048, 0.19531952],
[0.86529144, 0.13470856],
[0.75245951, 0.24754049],
[0.7527745 , 0.2472255 ],
[0.75403176, 0.24596824],
[0.12887064, 0.87112936],
[0.75245951, 0.24754049],
[0.27480928, 0.72519072],
[0.75270546, 0.24729454],
[0.80308014, 0.19691986],
[0.40535014, 0.59464986],
[0.28414073, 0.71585927],
[0.29900425, 0.70099575],
[0.29424034, 0.70575966],
[0.5635211 , 0.4364789 ],
[0.56643175, 0.43356825],
[0.75308922, 0.24691078],
[0.12887064, 0.87112936],
[0.23944262, 0.76055738],
[0.86548852, 0.13451148],
[0.91265361, 0.08734639],
[0.08800759, 0.91199241],
[0.34705733, 0.65294267],
[0.61207652, 0.38792348],
[0.40575801, 0.59424199],
[0.3970632 , 0.6029368 ],
[0.21411229, 0.78588771],
[0.15452766, 0.84547234],
[0.43678858, 0.56321142],
[0.73677655, 0.26322345],
[0.75371785, 0.24628215],
[0.8761031 , 0.1238969 ],
[0.19746072, 0.80253928],
[0.50237647, 0.49762353],
[0.75403176, 0.24596824],
[0.70847189, 0.29152811],
[0.75559726, 0.24440274],
[0.86489655, 0.13510345],
[0.30014706, 0.69985294],
[0.73644831, 0.26355169],
[0.39503985, 0.60496015],
[0.75559726, 0.24440274],
[0.75465877, 0.24534123],
[0.7060199 , 0.2939801 ],
[0.73169505, 0.26830495],
[0.40902552, 0.59097448],
[0.50634641, 0.49365359],
[0.70742251, 0.29257749],
[0.75270546, 0.24729454],
[0.56268867, 0.43731133]])
```

In [58]:
```
predicted_proba.shape
```

Out[58]: (140, 2)

In [59]:
```
# select ONLY the probabilities referring to 1s
predicted_proba[:,1]
```

Out[59]:
```
array([0.24691078, 0.39073909, 0.5140425 , 0.2447153 , 0.9160325 ,
       0.69807305, 0.69833226, 0.8848955 , 0.26224033, 0.24596824,
       0.49280785, 0.80280724, 0.93836804, 0.29082975, 0.70719453,
       0.4758953 , 0.49323071, 0.49111648, 0.632992  , 0.93644339,
       0.26355169, 0.2447153 , 0.52542844, 0.52711557, 0.77973465,
       0.26191315, 0.48815488, 0.12316421, 0.76554437, 0.2447153 ,
       0.38912926, 0.71585927, 0.70056321, 0.49365359, 0.2447153 ,
       0.59546518, 0.26256777, 0.78560289, 0.43689507, 0.60455576,
       0.24440274, 0.49889237, 0.26125957, 0.44701216, 0.80689602,
       0.60831063, 0.72248525, 0.24534123, 0.24855634, 0.24409045,
       0.50227596, 0.32586203, 0.69807305, 0.24697976, 0.82067559,
       0.39154489, 0.90719507, 0.26697852, 0.35791316, 0.35830205,
       0.70399998, 0.6991417 , 0.26797295, 0.78118404, 0.24619646,
       0.24659633, 0.09025875, 0.26289548, 0.7674276 , 0.29222745,
       0.26060705, 0.35283656, 0.88319236, 0.43606282, 0.593834  ,
       0.2447153 , 0.76584789, 0.74691753, 0.70276574, 0.63456269,
       0.26158623, 0.0876165 , 0.26996886, 0.71860418, 0.46633803,
       0.12261714, 0.69949744, 0.52711557, 0.2535535 , 0.71585927,
       0.19531952, 0.13470856, 0.24754049, 0.2472255 , 0.24596824,
       0.87112936, 0.24754049, 0.72519072, 0.24729454, 0.19691986,
       0.59464986, 0.71585927, 0.70099575, 0.70575966, 0.4364789 ,
       0.43356825, 0.24691078, 0.87112936, 0.76055738, 0.13451148,
       0.08734639, 0.91199241, 0.65294267, 0.38792348, 0.59424199,
       0.6029368 , 0.78588771, 0.84547234, 0.56321142, 0.26322345,
       0.24628215, 0.1238969 , 0.80253928, 0.49762353, 0.24596824,
       0.29152811, 0.24440274, 0.13510345, 0.69985294, 0.26355169,
       0.60496015, 0.24440274, 0.24534123, 0.2939801 , 0.26830495,
       0.59097448, 0.49365359, 0.29257749, 0.24729454, 0.43731133])
```

Save the model

In [60]:
```
# import the relevant module
import pickle
```

In [61]:
```
# pickle the model file
with open('model', 'wb') as file:
    pickle.dump(reg, file)
```

In [62]:
```
# pickle the scaler file
with open('scaler','wb') as file:
    pickle.dump(absenteeism_scaler, file)
```

In [ ]:
```

```

Dec 7, 2019 (http://www.datasciencelovers.com/2019/12/)

Absenteeism prediction Integration with Mysql-part-2 (http://www.datasciencelovers.com/machine-learning-projects/absenteeism-prediction-integration-with-mysql-part-2/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/absenteeism-prediction-integration-with-mysql-part-2/)**
Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/) Tag **absenteeism prediction (http://www.datasciencelovers.com/tag/absenteeism-prediction/)**, **logistic regression (http://www.datasciencelovers.com/tag/logistic-regression/)**, **python-sql-Tableau (http://www.datasciencelovers.com/tag/python-sql-tableau/)**

As we have already prepare the model now we need to import it in a new jupyter notebook file.

Predict the data for new data frame and store it into Mysql database. Import the library **pymysql** and and insert the new predicted data.

Lets see how we can do it in jupyter notebook.

Absenteesim Exercise - Integration

In [1]:
```
#Import previously saved model
from absenteeism_module import *
model = absenteeism_model('model', 'scaler')

#Import new data set
model.load_and_clean_data('Absenteeism_new_data.csv')

# Predict with new value
model.predicted_outputs()
```

Out[1]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month Value	Transportation Expense	Age	B M In
0	0	0.0	0	1	6	179	30	
1	1	0.0	0	0	6	361	28	
2	0	0.0	0	1	6	155	34	
3	0	0.0	0	1	6	179	40	
4	1	0.0	0	0	6	155	34	
5	1	0.0	0	0	6	225	28	
6	1	0.0	0	0	6	118	46	
7	0	0.0	0	1	6	179	30	
8	0	0.0	0	1	6	118	37	
9	1	0.0	0	0	6	118	37	
10	0	0.0	0	1	6	378	36	
11	0	0.0	1	0	6	118	50	
12	0	0.0	1	0	6	233	31	
13	0	0.0	0	1	6	179	30	
14	0	0.0	0	0	6	235	48	
15	0	0.0	0	0	6	268	33	
16	0	0.0	1	0	6	118	50	
17	1	0.0	0	0	6	179	30	
18	0	0.0	0	1	6	291	40	
19	1	0.0	0	0	7	179	30	
20	0	0.0	0	1	7	118	37	
21	0	0.0	0	1	7	233	31	
22	1	0.0	0	0	7	118	37	
23	1	0.0	0	0	7	118	37	
24	0	0.0	0	1	7	233	31	
25	0	0.0	0	1	7	235	43	
26	0	0.0	1	0	7	233	31	

27	1	0.0	0	0	7	228	58
28	0	0.0	0	1	7	118	37
29	1	0.0	0	0	7	228	58
30	0	0.0	0	1	7	189	33
31	0	0.0	0	1	7	118	37
32	0	0.0	0	1	7	361	28
33	0	0.0	0	1	7	225	28
34	1	0.0	0	0	7	369	31
35	1	0.0	0	0	7	289	33
36	1	0.0	0	0	7	235	37
37	0	0.0	0	0	7	118	40
38	0	0.0	0	0	7	231	39
39	0	0.0	0	0	7	179	53

Create Database cnnection

In [2]:
```
#Import library for mysql
import pymysql
```

In [3]:
```
# Specify the database details for further peocess.
conn = pymysql.connect(database = 'predicted_outputs', user = 'n
ativeuser', password = '365Pass')
```

In [4]:
```
#Create the cursor to involke the connection
cursor = conn.cursor()
```

Checkpoint 'df_new_obs'

In [5]:
```
#Store the predicted output in the nwe data frame called 'df_new
_obs'
df_new_obs = model.predicted_outputs()
df_new_obs
```

Out[5]:

	Reason_1	Reason_2	Reason_3	Reason_4	Month Value	Transportation Expense	Age	B M In
0	0	0.0	0	1	6	179	30	
1	1	0.0	0	0	6	361	28	
2	0	0.0	0	1	6	155	34	
3	0	0.0	0	1	6	179	40	
4	1	0.0	0	0	6	155	34	
5	1	0.0	0	0	6	225	28	
6	1	0.0	0	0	6	118	46	

7	0	0.0	0	1	6	179	30
8	0	0.0	0	1	6	118	37
9	1	0.0	0	0	6	118	37
10	0	0.0	0	1	6	378	36
11	0	0.0	1	0	6	118	50
12	0	0.0	1	0	6	233	31
13	0	0.0	0	1	6	179	30
14	0	0.0	0	0	6	235	48
15	0	0.0	0	0	6	268	33
16	0	0.0	1	0	6	118	50
17	1	0.0	0	0	6	179	30
18	0	0.0	0	1	6	291	40
19	1	0.0	0	0	7	179	30
20	0	0.0	0	1	7	118	37
21	0	0.0	0	1	7	233	31
22	1	0.0	0	0	7	118	37
23	1	0.0	0	0	7	118	37
24	0	0.0	0	1	7	233	31
25	0	0.0	0	1	7	235	43
26	0	0.0	1	0	7	233	31
27	1	0.0	0	0	7	228	58
28	0	0.0	0	1	7	118	37
29	1	0.0	0	0	7	228	58
30	0	0.0	0	1	7	189	33
31	0	0.0	0	1	7	118	37
32	0	0.0	0	1	7	361	28
33	0	0.0	0	1	7	225	28
34	1	0.0	0	0	7	369	31
35	1	0.0	0	0	7	289	33
36	1	0.0	0	0	7	235	37
37	0	0.0	0	0	7	118	40
38	0	0.0	0	0	7	231	39
39	0	0.0	0	0	7	179	53

.execute() method

In [9]:
```
# To use .execute() method retrive the data from data base
cursor.execute('SELECT * FROM predicted_outputs;')
```

```
# Output 0 shows there is no data present in the table
```

Out[9]: 0

Creating the INSERT Statement

In [10]:
```
#Asssign insert query to variable
insert_query = 'INSERT INTO predicted_outputs VALUES '
```

In [11]:
```
insert_query
```

Out[11]: 'INSERT INTO predicted_outputs VALUES '

In [12]:
```
df_new_obs.shape
```

Out[12]: (40, 13)

So in our new data freame we have 40 rows and 13 column

In [15]:
```
df_new_obs['Age']
```

Out[15]:
```
0     30
1     28
2     34
3     40
4     34
5     28
6     46
7     30
8     37
9     37
10    36
11    50
12    31
13    30
14    48
15    33
16    50
17    30
18    40
19    30
20    37
21    31
22    37
23    37
24    31
25    43
26    31
27    58
28    37
29    58
30    33
31    37
32    28
33    28
34    31
35    33
```

```
36    37
37    40
38    39
39    53
Name: Age, dtype: int64
```

In [17]:
```
df_new_obs[df_new_obs.columns.values[6]][0]
```

Out[17]:
```
30
```

Use for loop to insert the data into data base

In [18]:
```
#The first iteration we want to check every row in the dataframe
and then run insert command
for i in range(df_new_obs.shape[0]):
    insert_query += '('

    #Similarly we want to insert data in column. Here j represen
t column
    for j in range(df_new_obs.shape[1]):
        insert_query += str(df_new_obs[df_new_obs.columns.values
[j]][i]) + ', '

    insert_query = insert_query[:-2] + '), '
```

In [19]:
```
insert_query
```

Out[19]:
```
'INSERT INTO predicted_outputs VALUES (0, 0.0, 0, 1, 6, 179, 3
0, 19, 1, 0, 0, 0.1050748143291057, 0), (1, 0.0, 0, 0, 6, 361,
28, 27, 0, 1, 4, 0.810119898086104, 1), (0, 0.0, 0, 1, 6, 155,
34, 25, 0, 2, 0, 0.24117044969973478, 0), (0, 0.0, 0, 1, 6, 17
9, 40, 22, 1, 2, 0, 0.16072924168708813, 0), (1, 0.0, 0, 0, 6,
155, 34, 25, 0, 2, 0, 0.7514912643313632, 1), (1, 0.0, 0, 0, 6,
225, 28, 24, 0, 1, 2, 0.6929781448353303, 1), (1, 0.0, 0, 0, 6,
118, 46, 25, 0, 2, 0, 0.6329602375469388, 1), (0, 0.0, 0, 1, 6,
179, 30, 19, 1, 0, 0, 0.1050748143291057, 0), (0, 0.0, 0, 1, 6,
118, 37, 28, 0, 0, 0, 0.13391462868552698, 0), (1, 0.0, 0, 0,
6, 118, 37, 28, 0, 0, 0, 0.5953371990526135, 1), (0, 0.0, 0, 1,
6, 378, 36, 21, 0, 2, 4, 0.28610549839232, 0), (0, 0.0, 1, 0,
6, 118, 50, 31, 0, 1, 0, 0.7569925957613495, 1), (0, 0.0, 1, 0,
6, 233, 31, 21, 1, 1, 8, 0.22189357696108014, 0), (0, 0.0, 0,
1, 6, 179, 30, 19, 1, 0, 0, 0.1050748143291057, 0), (0, 0.0, 0,
0, 6, 235, 48, 33, 0, 1, 5, 0.0665590726801133, 0), (0, 0.0, 0,
0, 6, 268, 33, 25, 1, 0, 0, 0.18804298582412018, 0), (0, 0.0,
1, 0, 6, 118, 50, 31, 0, 1, 0, 0.7569925957613495, 1), (1, 0.0,
0, 0, 6, 179, 30, 19, 1, 0, 0, 0.5276685343597127, 1), (0, 0.0,
0, 1, 6, 291, 40, 25, 0, 1, 1, 0.32646901379851917, 0), (1, 0.
0, 0, 0, 7, 179, 30, 19, 1, 0, 0, 0.543491979581516, 1), (0, 0.
0, 0, 1, 7, 118, 37, 28, 0, 0, 0, 0.14146688138416358, 0), (0,
0.0, 0, 1, 7, 233, 31, 21, 1, 1, 8, 0.017511983341024603, 0),
(1, 0.0, 0, 0, 7, 118, 37, 28, 0, 0, 0, 0.6105667631385643, 1),
(1, 0.0, 0, 0, 7, 118, 37, 28, 0, 0, 0, 0.6105667631385643, 1),
(0, 0.0, 0, 1, 7, 233, 31, 21, 1, 1, 8, 0.017511983341024603,
0), (0, 0.0, 0, 1, 7, 235, 43, 38, 0, 1, 0, 0.4772856504385141,
0), (0, 0.0, 1, 0, 7, 233, 31, 21, 1, 1, 8, 0.2330722593383316
2, 0), (1, 0.0, 0, 0, 7, 228, 58, 22, 0, 2, 1, 0.68374854991907
92, 1), (0, 0.0, 0, 1, 7, 118, 37, 28, 0, 0, 0, 0.1414668813841
6358, 0), (1, 0.0, 0, 0, 7, 228, 58, 22, 0, 2, 1, 0.68374854991
90792, 1), (0, 0.0, 0, 1, 7, 189, 33, 25, 0, 2, 2, 0.1877515810
```

```
858458, 0), (0, 0.0, 0, 1, 7, 118, 37, 28, 0, 0, 0, 0.141466881
38416358, 0), (0, 0.0, 0, 1, 7, 361, 28, 27, 0, 1, 4, 0.3233444
855579362, 0), (0, 0.0, 0, 1, 7, 225, 28, 24, 0, 1, 2, 0.201788
42764749713, 0), (1, 0.0, 0, 0, 7, 369, 31, 25, 0, 3, 0, 0.9646
018697022684, 1), (1, 0.0, 0, 0, 7, 289, 33, 30, 0, 2, 1, 0.906
6189840747448, 1), (1, 0.0, 0, 0, 7, 235, 37, 29, 1, 1, 1, 0.73
18389674842187, 1), (0, 0.0, 0, 0, 7, 118, 40, 34, 0, 1, 8, 0.0
12369097946649985, 0), (0, 0.0, 0, 0, 7, 231, 39, 35, 0, 2, 2,
0.2666906934556679, 0), (0, 0.0, 0, 0, 7, 179, 53, 25, 0, 1, 1,
0.0960731564962269, 0), '
```

As we can see above insert query at the end it is , but in data base query it should be ;. To do that we need to follow below steps

In [20]:
```
#Remove ',' from insert query and pur ';' in that.
insert_query = insert_query[:-2] + ';'
```

In [21]:
```
# Now check again it has been corrected or not
insert_query
```

Out[21]:
```
'INSERT INTO predicted_outputs VALUES (0, 0.0, 0, 1, 6, 179, 3
0, 19, 1, 0, 0, 0.1050748143291057, 0), (1, 0.0, 0, 0, 6, 361,
28, 27, 0, 1, 4, 0.810119898086104, 1), (0, 0.0, 0, 1, 6, 155,
34, 25, 0, 2, 0, 0.24117044969973478, 0), (0, 0.0, 0, 1, 6, 17
9, 40, 22, 1, 2, 0, 0.16072924168708813, 0), (1, 0.0, 0, 0, 6,
155, 34, 25, 0, 2, 0, 0.7514912643313632, 1), (1, 0.0, 0, 0, 6,
225, 28, 24, 0, 1, 2, 0.6929781448353303, 1), (1, 0.0, 0, 0, 6,
118, 46, 25, 0, 2, 0, 0.6329602375469388, 1), (0, 0.0, 0, 1, 6,
179, 30, 19, 1, 0, 0, 0.1050748143291057, 0), (0, 0.0, 0, 1, 6,
118, 37, 28, 0, 0, 0, 0.13391462868552698, 0), (1, 0.0, 0, 0,
6, 118, 37, 28, 0, 0, 0, 0.5953371990526135, 1), (0, 0.0, 0, 1,
6, 378, 36, 21, 0, 2, 4, 0.28610549839232, 0), (0, 0.0, 1, 0,
6, 118, 50, 31, 0, 1, 0, 0.7569925957613495, 1), (0, 0.0, 1, 0,
6, 233, 31, 21, 1, 1, 8, 0.22189357696108014, 0), (0, 0.0, 0,
1, 6, 179, 30, 19, 1, 0, 0, 0.1050748143291057, 0), (0, 0.0, 0,
0, 6, 235, 48, 33, 0, 1, 5, 0.0665590726801133, 0), (0, 0.0, 0,
0, 6, 268, 33, 25, 1, 0, 0, 0.18804298582412018, 0), (0, 0.0,
1, 0, 6, 118, 50, 31, 0, 1, 0, 0.7569925957613495, 1), (1, 0.0,
0, 0, 6, 179, 30, 19, 1, 0, 0, 0.5276685343597127, 1), (0, 0.0,
0, 1, 6, 291, 40, 25, 0, 1, 1, 0.32646901379851917, 0), (1, 0.
0, 0, 0, 7, 179, 30, 19, 1, 0, 0, 0.543491979581516, 1), (0, 0.
0, 0, 1, 7, 118, 37, 28, 0, 0, 0, 0.14146688138416358, 0), (0,
0.0, 0, 1, 7, 233, 31, 21, 1, 1, 8, 0.017511983341024603, 0),
(1, 0.0, 0, 0, 7, 118, 37, 28, 0, 0, 0, 0.6105667631385643, 1),
(1, 0.0, 0, 0, 7, 118, 37, 28, 0, 0, 0, 0.6105667631385643, 1),
(0, 0.0, 0, 1, 7, 233, 31, 21, 1, 1, 8, 0.017511983341024603,
0), (0, 0.0, 0, 1, 7, 235, 43, 38, 0, 1, 0, 0.4772856504385141,
0), (0, 0.0, 1, 0, 7, 233, 31, 21, 1, 1, 8, 0.2330722593383316
2, 0), (1, 0.0, 0, 0, 7, 228, 58, 22, 0, 2, 1, 0.68374854991907
92, 1), (0, 0.0, 0, 1, 7, 118, 37, 28, 0, 0, 0, 0.1414668813841
6358, 0), (1, 0.0, 0, 0, 7, 228, 58, 22, 0, 2, 1, 0.68374854991
90792, 1), (0, 0.0, 0, 1, 7, 189, 33, 25, 0, 2, 2, 0.1877515810
858458, 0), (0, 0.0, 0, 1, 7, 118, 37, 28, 0, 0, 0, 0.141466881
38416358, 0), (0, 0.0, 0, 1, 7, 361, 28, 27, 0, 1, 4, 0.3233444
855579362, 0), (0, 0.0, 0, 1, 7, 225, 28, 24, 0, 1, 2, 0.201788
42764749713, 0), (1, 0.0, 0, 0, 7, 369, 31, 25, 0, 3, 0, 0.9646
018697022684, 1), (1, 0.0, 0, 0, 7, 289, 33, 30, 0, 2, 1, 0.906
6189840747448, 1), (1, 0.0, 0, 0, 7, 235, 37, 29, 1, 1, 1, 0.73
18389674842187, 1), (0, 0.0, 0, 0, 7, 118, 40, 34, 0, 1, 8, 0.0
```

```
12369097946649985, 0), (0, 0.0, 0, 0, 7, 231, 39, 35, 0, 2, 2,
```

Nov 10, 2019 (http://www.datasciencelovers.com/2019/11/)

Banking Credit Card Spend Prediction and Identify Drivers for Spends (http://www.datasciencelovers.com/machine-learning-projects/banking-credit-card-spend-prediction-and-identify-drivers-for-spends/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/banking-credit-card-spend-prediction-and-identify-drivers-for-spends/)**
Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/) Tag **credit card spend prediction (http://www.datasciencelovers.com/tag/credit-card-spend-prediction/)**, **linear regression (http://www.datasciencelovers.com/tag/linear-regression/)**, **machine learning (http://www.datasciencelovers.com/tag/machine-learning/)**, **regression (http://www.datasciencelovers.com/tag/regression/)**, **supervised learning (http://www.datasciencelovers.com/tag/supervised-learning/)**

Business Problem:

One of the global banks would like to understand what factors driving credit card spend are. The bank want use these insights to calculate credit limit. In order to solve the problem, the bank conducted survey of 5000 customers and collected data.

The objective of this case study is to understand what's driving the total spend (Primary Card + Secondary card). Given the factors, predict credit limit for the new applicants.

Data Availability:

- Data for the case are available in xlsx format.
- The data have been provided for 5000 customers.
- Detailed data dictionary has been provided for understanding the data in the data.
- Data is encoded in the numerical format to reduce the size of the data however some of the variables are categorical. You can find the details in the data dictionary

Let's develop a machine learning model for further analysis.

Machine learning work steps through python

Import all useable library

In [1]:
```
# import relevant modules
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as sn
import scipy.stats as stats
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from statsmodels.stats.outliers_influence import variance_inflat
ion_factor
from patsy import dmatrices
%matplotlib inline

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Settings
pd.set_option('display.max_columns', None)
np.set_printoptions(threshold=np.nan)
np.set_printoptions(precision=3)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

Load the Data set.

In [2]:
```
custdata_df = pd.read_excel("Data Set.xlsx", sheetname="customer
_dbase")
```

In [3]:
```
custdata_df.sample(5)
```

Out[3]:

	custid	region	townsize	gender	age	agecat	birthmonth	ed	edcat
394	8512-KZJXAA-A34	5	2.0	0	58	5	July	13	2
218	8477-FURXBL-V98	1	1.0	0	47	4	March	15	3
4015	0409-MMPGJY-ECA	1	1.0	1	60	5	January	12	2
	9411-								

4407	CNRRPX-2HW	3	2.0	0	56	5	July	17	4
1044	1890-WDOXSL-5H1	2	3.0	0	20	2	May	10	1

In [4]:
```
# Find column information in the dataframe.
custdata_df.columns
```

Out[4]:
```
Index(['custid', 'region', 'townsize', 'gender', 'age', 'ageca
t', 'birthmonth',
       'ed', 'edcat', 'jobcat',
       ...
       'owncd', 'ownpda', 'ownpc', 'ownipod', 'owngame', 'ownfa
x', 'news',
       'response_01', 'response_02', 'response_03'],
      dtype='object', length=130)
```

Creating Dependent Y column

In [5]:
```
#To create Y we need to sumup cardspent(first card spent amount)
and card2spent(Second card spent amount)
custdata_df['totalspend'] = custdata_df['cardspent'] + custdata_
df['card2spent']
```

In [6]:
```
custdata_df.head()
```

Out[6]:

	custid	region	townsize	gender	age	agecat	birthmonth	ed	edcat	jol
0	3964-QJWTRG-NPN	1	2.0	1	20	2	September	15	3	
1	0648-AIPJSP-UVM	5	5.0	0	22	2	May	17	4	
2	5195-TLUDJE-HVO	3	4.0	1	67	6	June	14	2	
3	4459-VLPQUH-3OL	4	3.0	0	23	2	May	16	3	
4	8158-SMTQFB-CNO	2	2.0	0	26	3	July	16	3	

In [7]:
```
# Now Run pandas profiling to see the data audit reports

import pandas_profiling
pandas_profiling.ProfileReport(custdata_df)
```

Out[7]:

Overview

Dataset info

Number of variables	131
Number of observations	5000
Total Missing (%)	0.2%
Total size in memory	5.0 MiB
Average record size in memory	1.0 KiB

Variables types

Numeric	59
Categorical	1
Boolean	49
Date	0
Text (Unique)	1
Rejected	21
Unsupported	0

Warnings

- address has 245 / 4.9% zeros Zeros
- addresscat is highly correlated with address (ρ = 0.92352) Rejected
- agecat is highly correlated with age (ρ = 0.96988) Rejected
- carbought has 2901 / 58.0% zeros Zeros
- card2spent has 179 / 3.6% zeros Zeros
- card2tenure is highly correlated with cardtenure (ρ = 0.96298) Rejected
- card2tenurecat is highly correlated with card2tenure (ρ = 0.92439) Rejected
- cardmon has 1419 / 28.4% zeros Zeros
- cardten has 1420 / 28.4% zeros Zeros
- cardtenure has 91 / 1.8% zeros Zeros
- carown has 799 / 16.0% zeros Zeros
- cars has 497 / 9.9% zeros Zeros
- cartype has 2287 / 45.7% zeros Zeros
- commutecat is highly correlated with commute (ρ = 0.98117) Rejected
- edcat is highly correlated with ed (ρ = 0.9639) Rejected
- employ has 659 / 13.2% zeros Zeros
- equipmon is highly correlated with equip (ρ = 0.94051) Rejected
- equipten has 3296 / 65.9% zeros Zeros
- hourstv has 85 / 1.7% zeros Zeros
- inccat is highly correlated with lninc (ρ = 0.94879) Rejected
- internet has 2498 / 50.0% zeros Zeros
- lncardmon is highly correlated with cardmon (ρ = 0.91687) Rejected
- lncardten has 1422 / 28.4% missing values Missing
- lnequipmon is highly correlated with equipmon (ρ = 0.97931) Rejected
- lnequipten is highly correlated with lntollten (ρ = 0.96611) Rejected
- lnlongten is highly correlated with lnlongmon (ρ = 0.92171)

Rejected

- lntollmon is highly correlated with tollmon (ρ = 0.93783) Rejected
- lntollten is highly correlated with lnlongten (ρ = 0.93139) Rejected
- lnwiremon is highly correlated with wiremon (ρ = 0.95389) Rejected
- lnwireten is highly correlated with lnequipten (ρ = 0.98318) Rejected
- longten is highly correlated with longmon (ρ = 0.9857) Rejected
- pets has 1529 / 30.6% zeros Zeros
- pets_birds has 4698 / 94.0% zeros Zeros
- pets_cats has 3413 / 68.3% zeros Zeros
- pets_dogs has 3762 / 75.2% zeros Zeros
- pets_freshfish has 3462 / 69.2% zeros Zeros
- pets_reptiles has 4818 / 96.4% zeros Zeros
- pets_saltfish has 4942 / 98.8% zeros Zeros
- pets_small has 4749 / 95.0% zeros Zeros
- spoused is highly correlated with marital (ρ = 0.95577) Rejected
- spousedcat is highly correlated with spoused (ρ = 0.98403) Rejected
- tenure is highly correlated with card2tenure (ρ = 0.92824) Rejected
- tollmon has 2622 / 52.4% zeros Zeros
- tollten has 2622 / 52.4% zeros Zeros
- totalspend is highly correlated with cardspent (ρ = 0.94149) Rejected
- wiremon has 3656 / 73.1% zeros Zeros
- wireten has 3656 / 73.1% zeros Zeros

Variables

active

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.466

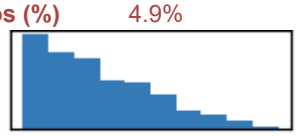
0	2670
1	2330

Toggle details

address

Numeric

Distinct count	57
Unique (%)	1.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	16.402
Minimum	0
Maximum	57
Zeros (%)	4.9%



Toggle details

~~addresscat~~

Highly correlated

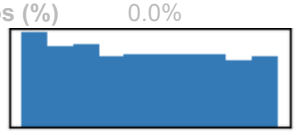
This variable is highly correlated with `address` *and should be ignored for analysis*

Correlation	0.92352

age

Numeric

Distinct count	62
Unique (%)	1.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	47.026
Minimum	18
Maximum	79
Zeros (%)	0.0%



Toggle details

~~agecat~~

Highly correlated

This variable is highly correlated with `age` *and should be ignored for analysis*

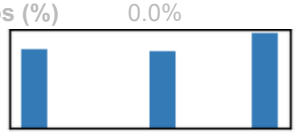
Correlation	0.96988

bfast

blast

Numeric

Distinct count	3
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.0586
Minimum	1
Maximum	3
Zeros (%)	0.0%



Toggle details

birthmonth

Categorical

Distinct count	12
Unique (%)	0.2%
Missing (%)	0.0%
Missing (n)	0

September	458
May	451
January	420
Other values (9)	3671

Toggle details

callcard

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.7162

1	3581
0	1419

Toggle details

callid

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.4752

0	2624

1	2376

Toggle details

callwait

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.479

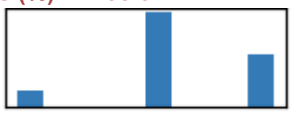
0	2605
1	2395

Toggle details

carbought

Numeric

Distinct count	3
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.221
Minimum	-1
Maximum	1
Zeros (%)	58.0%



Toggle details

carbuy

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.361

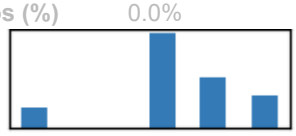
0	3195
1	1805

Toggle details

carcatvalue

Numeric

Distinct count	4
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	1.3894
Minimum	-1
Maximum	3
Zeros (%)	0.0%

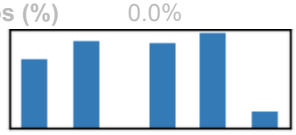


Toggle details

card

Numeric

Distinct count	5
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.7142
Minimum	1
Maximum	5
Zeros (%)	0.0%

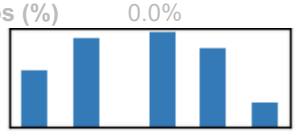


Toggle details

card2

Numeric

Distinct count	5
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.7744
Minimum	1
Maximum	5
Zeros (%)	0.0%

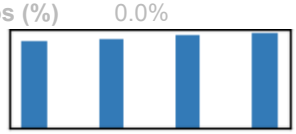


Toggle details

card2benefit

Numeric

Distinct count	4
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.534
Minimum	1
Maximum	4
Zeros (%)	0.0%



Toggle details

card2fee

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1872

0	4064
1	936

Toggle details

card2spent

Numeric

Distinct count	4477
Unique (%)	89.5%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	160.88
Minimum	0
Maximum	2069.2
Zeros (%)	3.6%

Toggle details

~~card2tenure~~

Highly correlated

This variable is highly correlated with `cardtenure` *and should be ignored for analysis*

Correlation	0.96298

~~card2tenurecat~~

Highly correlated

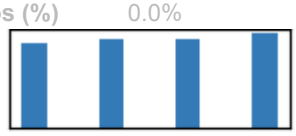
This variable is highly correlated with `card2tenure` *and should be ignored for analysis*

Correlation	0.92439

card2type

Numeric

Distinct count	4
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.5412
Minimum	1
Maximum	4
Zeros (%)	0.0%

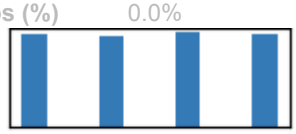


Toggle details

cardbenefit

Numeric

Distinct count	4
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.5058
Minimum	1
Maximum	4
Zeros (%)	0.0%



Toggle details

cardfee

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1898

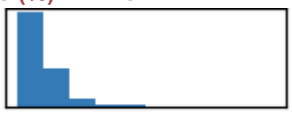
0	4051
1	949

Toggle details

cardmon

Numeric

Distinct count	271
Unique (%)	5.4%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	15.444
Minimum	0
Maximum	188.5
Zeros (%)	28.4%

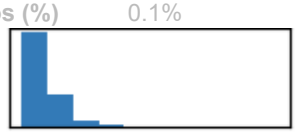


Toggle details

cardspent

Numeric

Distinct count	4760
Unique (%)	95.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	337.2
Minimum	0
Maximum	3926.4
Zeros (%)	0.1%



Toggle details

cardten

Numeric

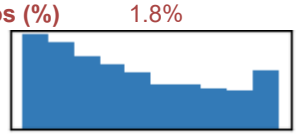
Distinct count	698
Unique (%)	14.0%
Missing (%)	0.0%
Missing (n)	2
Infinite (%)	0.0%
Infinite (n)	0
Mean	720.48
Minimum	0
Maximum	13705
Zeros (%)	28.4%

Toggle details

cardtenure

Numeric

Distinct count	41
Unique (%)	0.8%
Missing (%)	0.0%
Missing (n)	0
Infinfinite (%)	0.0%
Infinite (n)	0
Mean	16.656
Minimum	0
Maximum	40
Zeros (%)	1.8%

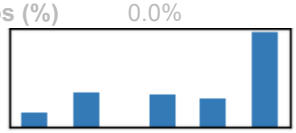


Toggle details

cardtenurecat

Numeric

Distinct count	5
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.7822
Minimum	1
Maximum	5
Zeros (%)	0.0%



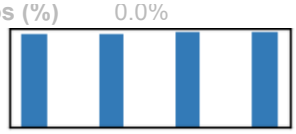
Toggle details

cardtype

Numeric

Distinct count	4
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.507
Minimum	1
Maximum	4

Zeros (%)	0.0%

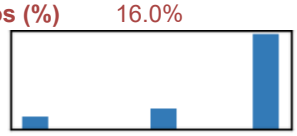


Toggle details

carown

Numeric

Distinct count	3
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.6414
Minimum	-1
Maximum	1
Zeros (%)	16.0%

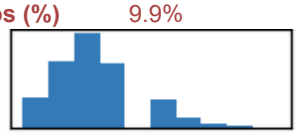


Toggle details

cars

Numeric

Distinct count	9
Unique (%)	0.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.1306
Minimum	0
Maximum	8
Zeros (%)	9.9%



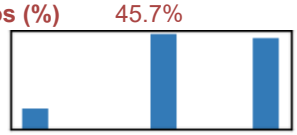
Toggle details

cartype

Numeric

Distinct count	3
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.3438
Minimum	-1
Maximum	1

Maximum	1
Zeros (%)	45.7%

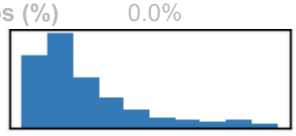


Toggle details

carvalue

Numeric

Distinct count	801
Unique (%)	16.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	23.233
Minimum	-1
Maximum	99.6
Zeros (%)	0.0%



Toggle details

churn

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.2532

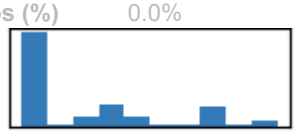
0	3734
1	1266

Toggle details

commute

Numeric

Distinct count	10
Unique (%)	0.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.9962
Minimum	1
Maximum	10
Zeros (%)	0.0%



Toggle details

commutebike

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1234

0	4383
1	617

Toggle details

commutebus

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.406

0	2970
1	2030

Toggle details

commutecar

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.679

1	3395
0	1605

Toggle details

commutecarpool

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.2718

0	3641

1	1359

Toggle details

~~commutecat~~

Highly correlated

This variable is highly correlated with `commute` *and should be ignored for analysis*

Correlation 0.98117

commutemotorcycle

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1026

0	4487
1	513

Toggle details

commutenonmotor

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.0584

0	4708
1	292

Toggle details

commutepublic

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.0954

0	4523
1	477

Toggle details

commuterail

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.2746

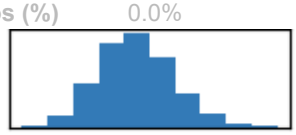
0	3627
1	1373

Toggle details

commutetime

Numeric

Distinct count	42
Unique (%)	0.8%
Missing (%)	0.0%
Missing (n)	2
Infinite (%)	0.0%
Infinite (n)	0
Mean	25.346
Minimum	8
Maximum	48
Zeros (%)	0.0%



Toggle details

commutewalk

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.3838

0	3081
1	1919

Toggle details

confer

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.478

0	2610

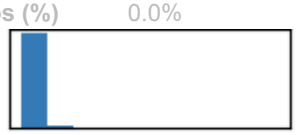
1	2390

Toggle details

creddebt

Numeric

Distinct count	4950
Unique (%)	99.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	1.8573
Minimum	0
Maximum	109.07
Zeros (%)	0.0%



Toggle details

custid

Categorical, Unique

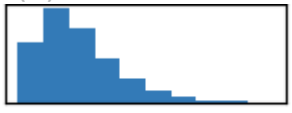
First 3 values
0394-AVUMJX-JAH
0191-VKRHCM-922
9844-XTDOZB-DSM
Last 3 values
6015-ASUOWY-VXJ
7186-WTDJGD-F2K
6289-YVKMBB-CXK

Toggle details

debtinc

Numeric

Distinct count	325
Unique (%)	6.5%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	9.9542
Minimum	0
Maximum	43.1
Zeros (%)	0.0%



Toggle details

default

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.2342

0	3829
1	1171

Toggle details

ebill

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.3486

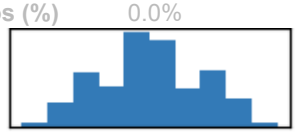
0	3257
1	1743

Toggle details

ed

Numeric

Distinct count	18
Unique (%)	0.4%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	14.543
Minimum	6
Maximum	23
Zeros (%)	0.0%



Toggle details

~~edcat~~

Highly correlated

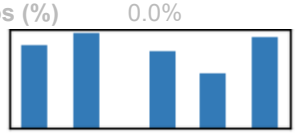
This variable is highly correlated with ed and should be ignored for analysis

Correlation 0.9639

empcat

Numeric

Distinct count	5
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.9326
Minimum	1
Maximum	5
Zeros (%)	0.0%

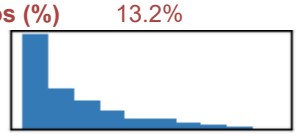


Toggle details

employ

Numeric

Distinct count	52
Unique (%)	1.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	9.7304
Minimum	0
Maximum	52
Zeros (%)	13.2%



Toggle details

equip

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.3408

0	3296
1	1704

Toggle details

~~equipmon~~

Highly correlated

This variable is highly correlated with `equip` *and should be ignored for*

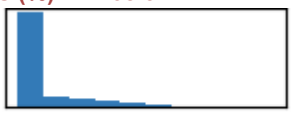
analysis

Correlation	0.94051

equipten

Numeric

Distinct count	1683
Unique (%)	33.7%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	470.18
Minimum	0
Maximum	6525.3
Zeros (%)	65.9%



Toggle details

forward

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.4806

0	2597
1	2403

Toggle details

gender

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.5036

1	2518
0	2482

Toggle details

homeown

Boolean

Distinct count	2
Unique (%)	0.0%

Missing (%)	0.0%
Missing (n)	0
Mean	0.6296

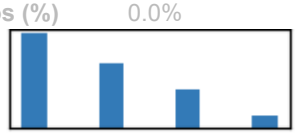
1	3148
0	1852

Toggle details

hometype

Numeric

Distinct count	4
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	1.8426
Minimum	1
Maximum	4
Zeros (%)	0.0%

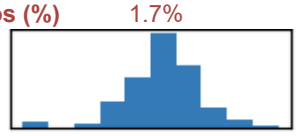


Toggle details

hourstv

Numeric

Distinct count	32
Unique (%)	0.6%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	19.645
Minimum	0
Maximum	36
Zeros (%)	1.7%



Toggle details

~~inccat~~

Highly correlated

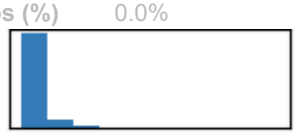
This variable is highly correlated with `Lninc` *and should be ignored for analysis*

Correlation	0.94879

income

Numeric

Distinct count	266
Unique (%)	5.3%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	54.76
Minimum	9
Maximum	1073
Zeros (%)	0.0%

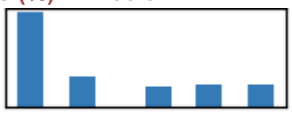


Toggle details

internet

Numeric

Distinct count	5
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	1.1996
Minimum	0
Maximum	4
Zeros (%)	50.0%

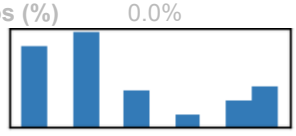


Toggle details

jobcat

Numeric

Distinct count	6
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.7528
Minimum	1
Maximum	6
Zeros (%)	0.0%

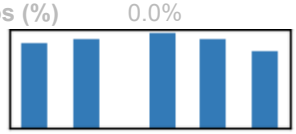


Toggle details

jobsat

Numeric

Distinct count	5
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.9642
Minimum	1
Maximum	5
Zeros (%)	0.0%



Toggle details

~~lncardmon~~

Highly correlated

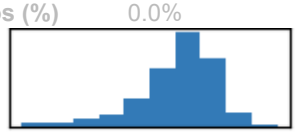
This variable is highly correlated with `cardmon` *and should be ignored for analysis*

Correlation	0.91687

lncardten

Numeric

Distinct count	697
Unique (%)	13.9%
Missing (%)	28.4%
Missing (n)	1422
Infinite (%)	0.0%
Infinite (n)	0
Mean	6.4263
Minimum	1.5581
Maximum	9.5255
Zeros (%)	0.0%



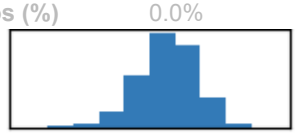
Toggle details

lncreddebt

Numeric

Distinct count	4942
Unique (%)	98.8%
Missing (%)	0.0%
Missing (n)	1
Infinite (%)	0.0%
Infinite (n)	0
Mean	-0.13045

Minimum	-6.5973
Maximum	4.692
Zeros (%)	0.0%



Toggle details

~~lnequipmon~~

Highly correlated

This variable is highly correlated with `equipmon` and should be ignored for analysis

Correlation	0.97931

~~lnequipten~~

Highly correlated

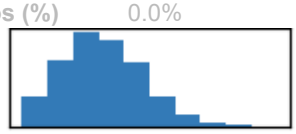
This variable is highly correlated with `lntollten` and should be ignored for analysis

Correlation	0.96611

lninc

Numeric

Distinct count	266
Unique (%)	5.3%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.6999
Minimum	2.1972
Maximum	6.9782
Zeros (%)	0.0%



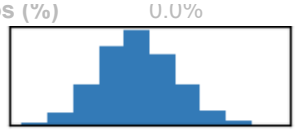
Toggle details

lnlongmon

Numeric

Distinct count	866
Unique (%)	17.3%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.2888
Minimum	-0.10536
Maximum	5.1921

Zeros (%) 0.0%



Toggle details

~~lnlongten~~

Highly correlated

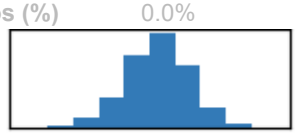
This variable is highly correlated with `lnlongmon` *and should be ignored for analysis*

Correlation 0.92171

lnothdebt

Numeric

Distinct count	4973
Unique (%)	99.5%
Missing (%)	0.0%
Missing (n)	1
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.69692
Minimum	-4.0921
Maximum	4.952
Zeros (%)	0.0%



Toggle details

~~lntollmon~~

Highly correlated

This variable is highly correlated with `tollmon` *and should be ignored for analysis*

Correlation 0.93783

~~lntollten~~

Highly correlated

This variable is highly correlated with `lnlongten` *and should be ignored for analysis*

Correlation 0.93139

~~lnwiremon~~

Highly correlated

This variable is highly correlated with `wiremon` *and should be ignored for*

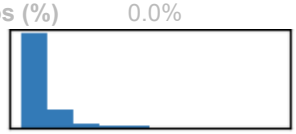
analysis
Correlation 0.95389

~~lnwireten~~
Highly correlated

This variable is highly correlated with `lnequipten` *and should be ignored for analysis*
Correlation 0.98318

longmon
Numeric

Distinct count	866
Unique (%)	17.3%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	13.471
Minimum	0.9
Maximum	179.85
Zeros (%)	0.0%



Toggle details

~~longten~~
Highly correlated

This variable is highly correlated with `Longmon` *and should be ignored for analysis*
Correlation 0.9857

marital
Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.4802

0	2599
1	2401

Toggle details

multline
Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.4884

0	2558
1	2442

Toggle details

news

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.4726

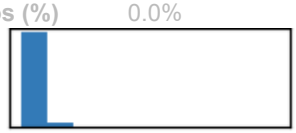
0	2637
1	2363

Toggle details

othdebt

Numeric

Distinct count	4973
Unique (%)	99.5%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.6545
Minimum	0
Maximum	141.46
Zeros (%)	0.0%



Toggle details

owncd

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.9328

1	4664
0	336

Toggle details

owndvd

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.9136

1	4568
0	432

Toggle details

ownfax

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1788

0	4106
1	894

Toggle details

owngame

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.4748

0	2626
1	2374

Toggle details

ownipod

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.4792

0	2604
1	2396

Toggle details

ownpc

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.6328

Toggle details

ownpda

Boolean

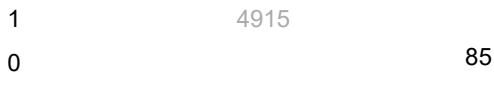
Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.201

Toggle details

owntv

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.983

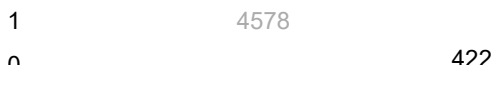


Toggle details

ownvcr

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.9156



Toggle details

pager

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.2436

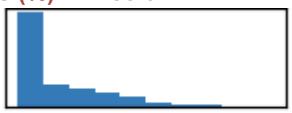
0	3782
1	1218

Toggle details

pets

Numeric

Distinct count	20
Unique (%)	0.4%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.0674
Minimum	0
Maximum	21
Zeros (%)	30.6%

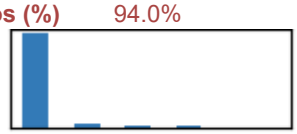


Toggle details

pets_birds

Numeric

Distinct count	6
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.1104
Minimum	0
Maximum	5
Zeros (%)	94.0%

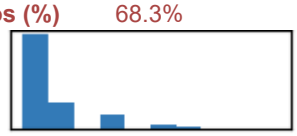


Toggle details

pets_cats

Numeric

Distinct count	7
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.5004
Minimum	0
Maximum	6
Zeros (%)	68.3%

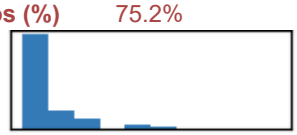


Toggle details

pets_dogs

Numeric

Distinct count	7
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.3924
Minimum	0
Maximum	7
Zeros (%)	75.2%

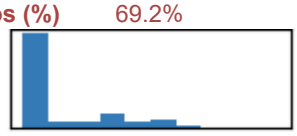


Toggle details

pets_freshfish

Numeric

Distinct count	17
Unique (%)	0.3%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	1.8474
Minimum	0
Maximum	16
Zeros (%)	69.2%



Toggle details

pets_reptiles

Numeric

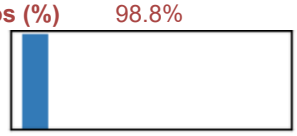
Distinct count	7
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.0556
Minimum	0
Maximum	6
Zeros (%)	96.4%

Toggle details

pets_saltfish

Numeric

Distinct count	9
Unique (%)	0.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.0466
Minimum	0
Maximum	8
Zeros (%)	98.8%

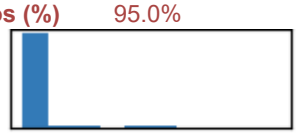


Toggle details

pets_small

Numeric

Distinct count	8
Unique (%)	0.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	0.1146
Minimum	0
Maximum	7
Zeros (%)	95.0%



Toggle details

polcontrib

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.2384

0	3808
1	1192

Toggle details

polparty

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.3814

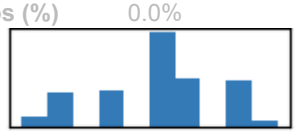
0	3093
1	1907

Toggle details

polview

Numeric

Distinct count	7
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	4.0886
Minimum	1
Maximum	7
Zeros (%)	0.0%



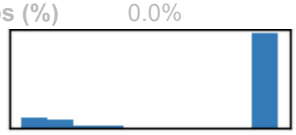
Toggle details

reason

Numeric

Distinct count	5
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0

Infinite (%)	0.0%
Infinite (n)	0
Mean	7.6368
Minimum	1
Maximum	9
Zeros (%)	0.0%

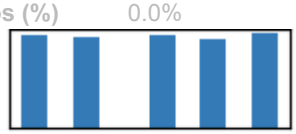


Toggle details

region

Numeric

Distinct count	5
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.0014
Minimum	1
Maximum	5
Zeros (%)	0.0%

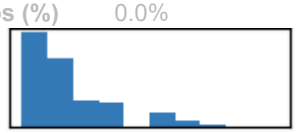


Toggle details

reside

Numeric

Distinct count	9
Unique (%)	0.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.204
Minimum	1
Maximum	9
Zeros (%)	0.0%



Toggle details

response_01

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%

Missing (n)	0
Mean	0.0836

0	4582
1	418

Toggle details

response_02

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1298

0	4351
1	649

Toggle details

response_03

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1026

0	4487
1	513

Toggle details

retire

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1476

0	4262
1	738

Toggle details

~~spoused~~

Highly correlated

This variable is highly correlated with `marital` *and should be ignored for analysis*

analysis
Correlation 0.95577

~~spousedcat~~
Highly correlated

This variable is highly correlated with `spoused` *and should be ignored for analysis*
Correlation 0.98403

telecommute
Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.188

0	4060
1	940

Toggle details

~~tenure~~
Highly correlated

This variable is highly correlated with `card2tenure` *and should be ignored for analysis*
Correlation 0.92824

tollfree
Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.4756

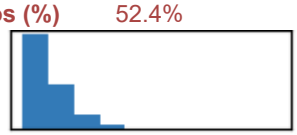
0	2622
1	2378

Toggle details

tollmon
Numeric

Distinct count	235
Unique (%)	4.7%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%

Infinite (n)	0
Mean	13.264
Minimum	0
Maximum	173
Zeros (%)	52.4%

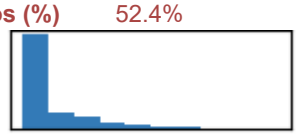


Toggle details

tollten

Numeric

Distinct count	2323
Unique (%)	46.5%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	577.83
Minimum	0
Maximum	6923.4
Zeros (%)	52.4%



Toggle details

~~totalspend~~

Highly correlated

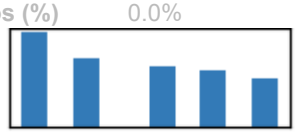
This variable is highly correlated with `cardspent` and should be ignored for analysis

Correlation	0.94149

townsize

Numeric

Distinct count	6
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	2
Infinite (%)	0.0%
Infinite (n)	0
Mean	2.6873
Minimum	1
Maximum	5
Zeros (%)	0.0%



Toggle details

union

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.1512

0	4244
1	756

Toggle details

voice

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.303

0	3485
1	1515

Toggle details

vote

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.518

1	2590
0	2410

Toggle details

wireless

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.2688

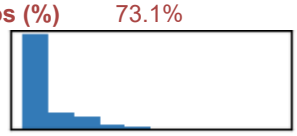
0	3656
1	1344

Toggle details

wiremon

Numeric

Distinct count	746
Unique (%)	14.9%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	10.701
Minimum	0
Maximum	186.25
Zeros (%)	73.1%

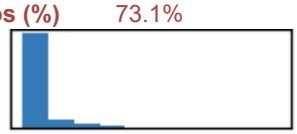


Toggle details

wireten

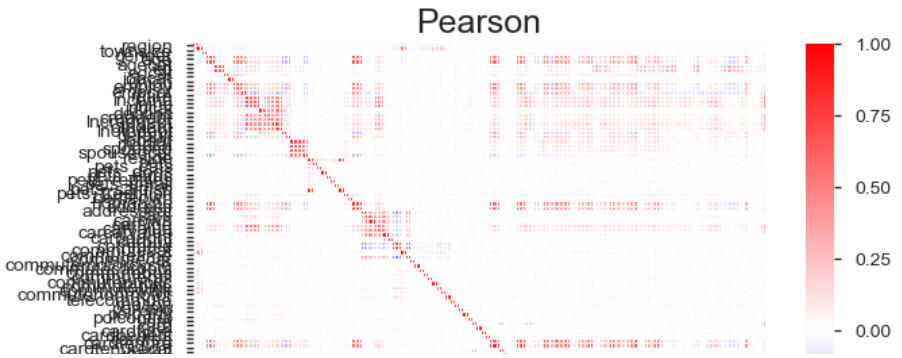
Numeric

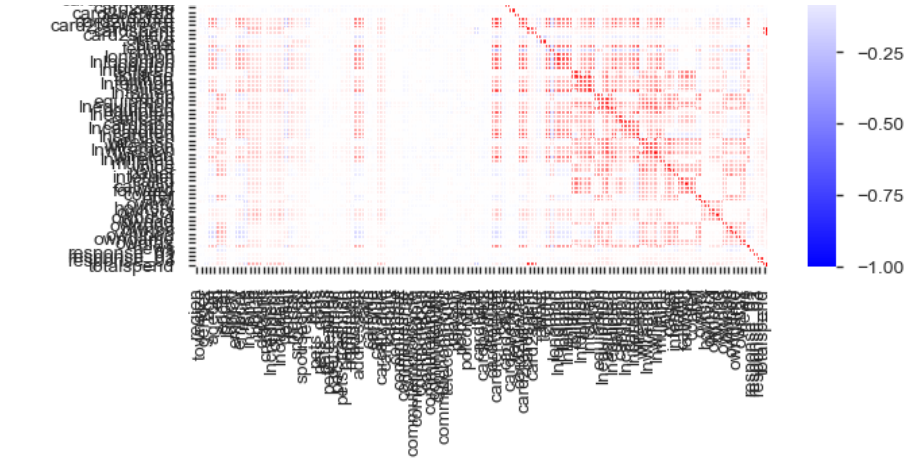
Distinct count	1328
Unique (%)	26.6%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	421.98
Minimum	0
Maximum	12859
Zeros (%)	73.1%

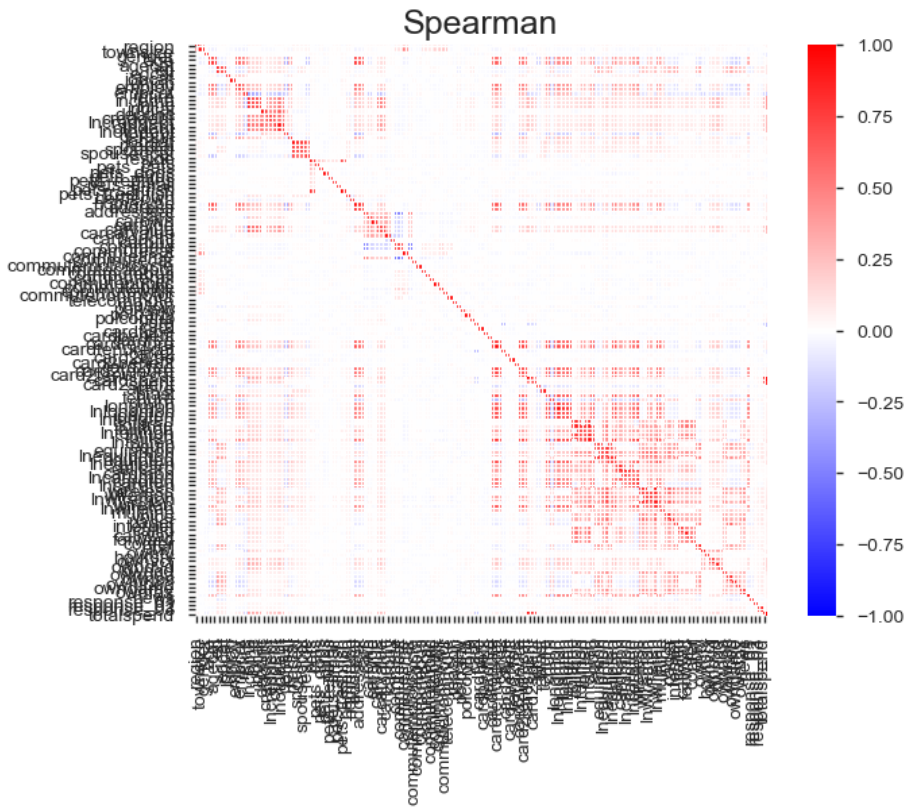


Toggle details

Correlations







Sample

	custid	region	townsize	gender	age	agecat	birthm
0	3964-QJWTRG-NPN	1	2.0	1	20	2	Septer
1	0648-AIPJSP-UVM	5	5.0	0	22	2	
2	5195-TLUDJE-HVO	3	4.0	1	67	6	
3	4459-VLPQUH-3OL	4	3.0	0	23	2	
4	8158-SMTQFB-CNO	2	2.0	0	26	3	

Drop Variables

In [7]:
```
#Drop cardspent(first card spent amount) and card2spent(Second c
ard spent amount) because its not adding any value
#Also  we are droping "custid","birthmonth" bacuse cust id is un
ique and birth month is not adding any value

custdata_df.drop(["cardspent","card2spent","custid"],axis=1, inp
lace=True)
```

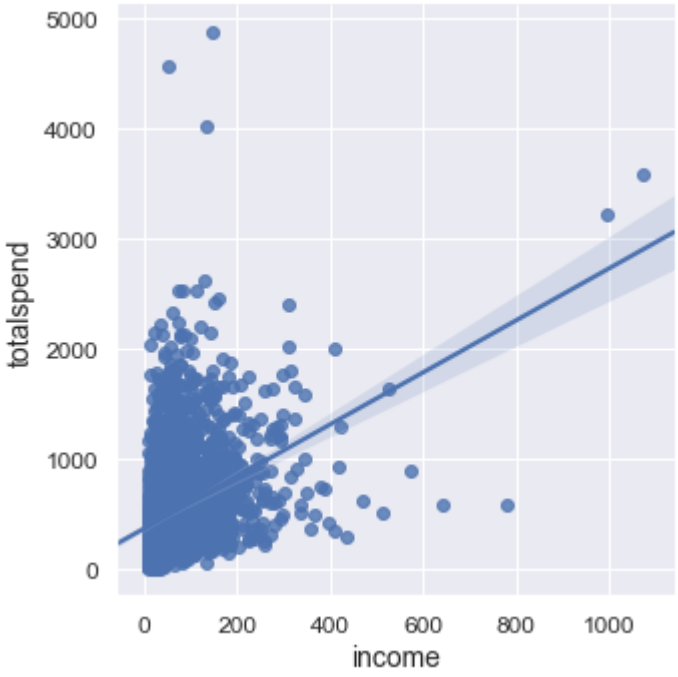
In [ ]:
```
#Item count need to drop promary and secondary Iem count
```

Check linearlity of data:

1. Use scatter plot by using 'df.plot' or if you want a linear line you can use 'sns.lmplot' through seaborn.

In [8]:
```
# Seaborn scatter plot with regression line
# aspect=1.5, scatter_kws={'alpha':0.2}) - You can choose this o
ption too.
sns.lmplot(x='income', y='totalspend', data=custdata_df)
```

Out[8]: <seaborn.axisgrid.FacetGrid at 0xba09c88>



So we can see above most data points are near to line which means our data is normal and we can go ahed for further process.

In [9]:
```
# Creating Categorical List= Contains categorical variables...

for x in ['region','townsize','gender','agecat','edcat','birthmo
nth','jobcat','union','employ','empcat','retire',
```

```
          'inccat','default','jobsat','marital','spousedcat','ho
meown','hometype','address','addresscat','cars','carown',
          'cartype','carcatvalue','carbought','carbuy','commut
e','commutecat','commutecar',
          'commutemotorcycle','commutecarpool','commutebus','com
muterail','commutepublic','commutebike','commutewalk',
         'commutenonmotor','telecommute','reason','polview','pol
party','polcontrib','vote','card','cardtype','cardbenefit',
          'cardfee','cardtenure','cardtenurecat','card2','card2ty
pe','card2benefit','card2fee','card2tenure','card2tenurecat',
          'active','bfast','churn','tollfree','equip','callcar
d','wireless','multline','voice','pager','internet','callid',
          'callwait','forward','confer','ebill','owntv','ownvc
r','owndvd','owncd','ownpda','ownpc','ownipod','owngame','ownfa
x',
        'news','response_01','response_02','response_03']:
    custdata_df[x]=custdata_df[x].astype('object')
```

In [ ]:

Separate numerical and categorical variable

To do that you need to apply for loop along with if condition.

In [10]:
```
# Find numerical variable in Data frame.
# This will return a list
numeric_var_names = [key for key in dict(custdata_df.dtypes) if
dict(custdata_df.dtypes)[key] in ['float64', 'int64', 'float32',
'int32']]

# Find Categorical variable in Data frame
cat_var_names = [key for key in dict(custdata_df.dtypes) if dict
(custdata_df.dtypes)[key] in ['object']]

#Print the data frame
print( numeric_var_names)
print(cat_var_names)
```

```
['age', 'ed', 'income', 'lninc', 'debtinc', 'creddebt', 'lncred
debt', 'othdebt', 'lnothdebt', 'spoused', 'reside', 'pets', 'pe
ts_cats', 'pets_dogs', 'pets_birds', 'pets_reptiles', 'pets_sma
ll', 'pets_saltfish', 'pets_freshfish', 'carvalue', 'commutetim
e', 'tenure', 'longmon', 'lnlongmon', 'longten', 'lnlongten',
'tollmon', 'lntollmon', 'tollten', 'lntollten', 'equipmon', 'ln
equipmon', 'equipten', 'lnequipten', 'cardmon', 'lncardmon', 'c
ardten', 'lncardten', 'wiremon', 'lnwiremon', 'wireten', 'lnwir
eten', 'hourstv', 'totalspend']
['region', 'townsize', 'gender', 'agecat', 'birthmonth', 'edca
t', 'jobcat', 'union', 'employ', 'empcat', 'retire', 'inccat',
'default', 'jobsat', 'marital', 'spousedcat', 'homeown', 'homet
ype', 'address', 'addresscat', 'cars', 'carown', 'cartype', 'ca
rcatvalue', 'carbought', 'carbuy', 'commute', 'commutecat', 'co
mmutecar', 'commutemotorcycle', 'commutecarpool', 'commutebus',
'commuterail', 'commutepublic', 'commutebike', 'commutewalk',
'commutenonmotor', 'telecommute', 'reason', 'polview', 'polpart
y', 'polcontrib', 'vote', 'card', 'cardtype', 'cardbenefit', 'c
ardfee', 'cardtenure', 'cardtenurecat', 'card2', 'card2type',
'card2benefit', 'card2fee', 'card2tenure', 'card2tenurecat', 'a
ctive', 'bfast', 'churn', 'tollfree', 'equip', 'callcard', 'wir
```

```
eless', 'multline', 'voice', 'pager', 'internet', 'callid', 'ca
llwait', 'forward', 'confer', 'ebill', 'owntv', 'ownvcr', 'ownd
vd', 'owncd', 'ownpda', 'ownpc', 'ownipod', 'owngame', 'ownfa
x', 'news', 'response_01', 'response_02', 'response_03']
```

In [11]:
```
#Convert a list in to dataframe
#Information about numericvariable in custdata_df

custdata_df_num=custdata_df[numeric_var_names]
custdata_df_num.head(5)
```

Out[11]:

	age	ed	income	lninc	debtinc	creddebt	lncreddebt	othdebt	lnothde
0	20	15	31	3.433987	11.1	1.200909	0.183079	2.240091	0.8065
1	22	17	15	2.708050	18.6	1.222020	0.200505	1.567980	0.4497
2	67	14	35	3.555348	9.9	0.928620	-0.074056	2.536380	0.9307
3	23	16	20	2.995732	5.7	0.022800	-3.780995	1.117200	0.1108
4	26	16	23	3.135494	1.7	0.214659	-1.538705	0.176341	-1.7353

In [12]:
```
#Convert a list in to dataframe
#Information about categorical variable in custdata_df

custdata_df_cat=custdata_df[cat_var_names]
custdata_df_cat.head(5)
```

Out[12]:

	region	townsize	gender	agecat	birthmonth	edcat	jobcat	union	employ
0	1	2	1	2	September	3	1	1	0
1	5	5	0	2	May	4	2	0	0
2	3	4	1	6	June	2	2	0	16
3	4	3	0	2	May	3	2	0	0
4	2	2	0	3	July	3	2	0	1

Creating Data audit Report

In [13]:
```
# Use a general function that returns multiple values
def var_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.sum(), x.me
an(), x.median(),  x.std(), x.var(), x.min(), x.dropna().quantil
e(0.01), x.dropna().quantile(0.05),x.dropna().quantile(0.10),x.d
ropna().quantile(0.25),x.dropna().quantile(0.50),x.dropna().quan
tile(0.75), x.dropna().quantile(0.90),x.dropna().quantile(0.95),
x.dropna().quantile(0.99),x.max()],
                  index=['N', 'NMISS', 'SUM', 'MEAN','MEDIAN',
'STD', 'VAR', 'MIN', 'P1' , 'P5' ,'P10' ,'P25' ,'P50' ,'P75' ,'P
90' ,'P95' ,'P99' ,'MAX'])

num_summary=custdata_df_num.apply(lambda x: var_summary(x)).T
```

In [14]:

In [14]:
```
num_summary
```

Out[14]:

	N	NMISS	SUM	MEAN	MEDIAN	S
age	5000.0	0.0	2.351280e+05	47.025600	47.000000	17.7703
ed	5000.0	0.0	7.271500e+04	14.543000	14.000000	3.2810
income	5000.0	0.0	2.737980e+05	54.759600	38.000000	55.3775
lninc	5000.0	0.0	1.849955e+04	3.699909	3.637586	0.7470
debtinc	5000.0	0.0	4.977080e+04	9.954160	8.800000	6.3997
creddebt	5000.0	0.0	9.286628e+03	1.857326	0.926437	3.4157
lncreddebt	4999.0	1.0	-6.521372e+02	-0.130454	-0.076106	1.2730
othdebt	5000.0	0.0	1.827230e+04	3.654460	2.098540	5.3951
lnothdebt	4999.0	1.0	3.483879e+03	0.696915	0.741537	1.1285
spoused	5000.0	0.0	3.056400e+04	6.112800	-1.000000	7.7435
reside	5000.0	0.0	1.102000e+04	2.204000	2.000000	1.3939
pets	5000.0	0.0	1.533700e+04	3.067400	2.000000	3.4144
pets_cats	5000.0	0.0	2.502000e+03	0.500400	0.000000	0.8607
pets_dogs	5000.0	0.0	1.962000e+03	0.392400	0.000000	0.7960
pets_birds	5000.0	0.0	5.520000e+02	0.110400	0.000000	0.4942
pets_reptiles	5000.0	0.0	2.780000e+02	0.055600	0.000000	0.3257
pets_small	5000.0	0.0	5.730000e+02	0.114600	0.000000	0.5687
pets_saltfish	5000.0	0.0	2.330000e+02	0.046600	0.000000	0.4695
pets_freshfish	5000.0	0.0	9.237000e+03	1.847400	0.000000	3.0748
carvalue	5000.0	0.0	1.161629e+05	23.232580	17.000000	21.2316
commutetime	4998.0	2.0	1.266770e+05	25.345538	25.000000	5.8791
tenure	5000.0	0.0	1.910240e+05	38.204800	38.000000	22.6618
longmon	5000.0	0.0	6.735725e+04	13.471450	9.550000	12.7733
lnlongmon	5000.0	0.0	1.144390e+04	2.288779	2.256541	0.7751
longten	4997.0	3.0	3.542232e+06	708.871753	350.000000	979.2910
lnlongten	4997.0	3.0	2.803966e+04	5.611298	5.857933	1.6493
tollmon	5000.0	0.0	6.632225e+04	13.264450	0.000000	16.3100
lntollmon	2378.0	2622.0	7.712400e+03	3.243230	3.228826	0.4046
tollten	5000.0	0.0	2.889163e+06	577.832510	0.000000	949.1515
lntollten	2378.0	2622.0	1.565861e+04	6.584783	6.858013	1.2220
equipmon	5000.0	0.0	6.495655e+04	12.991310	0.000000	19.2129
lnequipmon	1704.0	3296.0	6.134805e+03	3.600238	3.598681	0.2833
equipten	5000.0	0.0	2.350882e+06	470.176400	0.000000	912.2206
lnequipten	1704.0	3296.0	1.149739e+04	6.747296	7.050556	1.1992
cardmon	5000.0	0.0	7.721925e+04	15.443850	13.750000	15.0075
lncardmon	3581.0	1419.0	1.041975e+04	2.909733	2.904165	0.5648
cardten	4998.0	2.0	3.600951e+06	720.478391	425.000000	922.2255
lncardten	3578.0	1422.0	2.299333e+04	6.426309	6.639876	1.1720

wiremon	5000.0	0.0	5.350595e+04	10.701190	0.000000	19.7998
lnwiremon	1344.0	3656.0	4.845121e+03	3.605001	3.597997	0.3901
wireten	5000.0	0.0	2.109923e+06	421.984610	0.000000	1001.0032
lnwireten	1344.0	3656.0	9.150129e+03	6.808132	7.147185	1.2839
hourstv	5000.0	0.0	9.822500e+04	19.645000	20.000000	5.1656
totalspend	5000.0	0.0	2.490393e+06	498.078630	414.250000	351.5292

In [15]:
```
def cat_summary(x):
    return pd.Series([x.count(),x.isnull().sum(),x.value_counts
(),x.unique()],
                     index=['N','NMISS','ColumnNames','UniqueValu
es'])
```

In [16]:
```
cat_summary=custdata_df_cat.apply(lambda x:cat_summary(x)).T
```

In [17]:
```
cat_summary
```

Out[17]:

	N	NMISS	ColumnNames	UniqueValues
region	5000	0	5 1027 1 1009 3 1003 2 995 4 ...	[1, 5, 3, 4, 2]
townsize	4998	2	1.0 1436 2.0 1048 3.0 907 4.0 85...	[2.0, 5.0, 4.0, 3.0, 1.0, nan]
gender	5000	0	1 2518 0 2482 Name: gender, dtype: int64	[1, 0]
agecat	5000	0	4 1222 5 1195 6 1068 3 893 2 ...	[2, 6, 3, 5, 4]
birthmonth	5000	0	September 458 May 451 June ...	[September, May, June, July, August, October, ...
edcat	5000	0	2 1567 4 1111 3 1022 1 946 5 ...	[3, 4, 2, 1, 5]
jobcat	5000	0	2 1640 1 1388 6 688 3 620 5 ...	[1, 2, 3, 6, 4, 5]
union	5000	0	0 4244 1 756 Name: union, dtype: int64	[1, 0]
employ	5000	0	0 659 1 389 2 318 3 309 4 ...	[0, 16, 1, 22, 10, 11, 15, 19, 8, 4, 12, 3, 27...
empcat	5000	0	2 1180 5 1135 1 1048 3 968 4 ...	[1, 5, 3, 4, 2]
retire	5000	0	0 4262 1 738 Name: retire, dtype: int64	[0, 1]
inccat	5000	0	2 1797 1 1330 3 839 4 650 5 ...	[2, 1, 4, 3, 5]
default	5000	0	0 3829 1 1171 Name: default, dtype: int64	[1, 0]
jobsat	5000	0	3 1085 2 1031 4 1016 1 975 5 ...	[1, 4, 2, 5, 3]
marital	5000	0	0 2599 1 2401 Name: marital, dtype: int64	[0, 1]

spousedcat	5000	0	-1 2599 2 789 1 606 3 507 4...	[-1, 2, 4, 3, 1, 5]
homeown	5000	0	1 3148 0 1852 Name: homeown, dtype: int64	[0, 1]
hometype	5000	0	1 2265 2 1548 3 896 4 291 Name: ...	[2, 3, 1, 4]
address	5000	0	0 245 2 196 4 195 5 177 3 ...	[0, 2, 30, 3, 31, 21, 20, 19, 14, 5, 9, 32, 29...
addresscat	5000	0	3 1221 5 1157 4 1139 2 873 1 ...	[1, 5, 2, 4, 3]
cars	5000	0	2 1607 1 1119 3 1082 0 497 4 ...	[2, 3, 1, 0, 4, 5, 7, 6, 8]
carown	5000	0	1 3704 0 799 -1 497 Name: carown,...	[1, 0, -1]
cartype	5000	0	0 2287 1 2216 -1 497 Name: cartype...	[0, 1, -1]
carcatvalue	5000	0	1 2399 2 1267 3 837 -1 497 Na...	[1, -1, 2, 3]
carbought	5000	0	0 2901 1 1602 -1 497 Name: carboug...	[0, -1, 1]
carbuy	5000	0	0 3195 1 1805 Name: carbuy, dtype: int64	[0, 1]
commute	5000	0	1 2855 4 635 8 585 5 302 3 ...	[8, 1, 4, 6, 5, 3, 10, 2, 7, 9]
commutecat	5000	0	1 2905 3 981 4 666 2 295 5 ...	[4, 1, 3, 2, 5]
commutecar	5000	0	1 3395 0 1605 Name: commutecar, dtype: i...	[0, 1]
commutemotorcycle	5000	0	0 4487 1 513 Name: commutemotorcycle, d...	[1, 0]
...	...	...	...	...
card2tenurecat	5000	0	5 1923 2 1019 3 933 4 760 1 ...	[2, 5, 3, 1, 4]
active	5000	0	0 2670 1 2330 Name: active, dtype: int64	[0, 1]
bfast	5000	0	3 1875 1 1582 2 1543 Name: bfast, dty...	[3, 1, 2]
churn	5000	0	0 3734 1 1266 Name: churn, dtype: int64	[1, 0]
tollfree	5000	0	0 2622 1 2378 Name: tollfree, dtype: int64	[1, 0]
equip	5000	0	0 3296 1 1704 Name: equip, dtype: int64	[1, 0]
callcard	5000	0	1 3581 0 1419 Name: callcard, dtype: int64	[1, 0]
wireless	5000	0	0 3656 1 1344 Name: wireless, dtype: int64	[0, 1]
multline	5000	0	0 2558 1 2442 Name: multline, dtype: int64	[1, 0]
voice	5000	0	0 3485 1 1515 Name: voice, dtype: int64	[1, 0]

pager	5000	0	0 3782 1 1218 Name: pager, dtype: int64	[1, 0]
internet	5000	0	0 2498 1 774 3 598 4 585 2 ...	[0, 4, 2, 3, 1]
callid	5000	0	0 2624 1 2376 Name: callid, dtype: int64	[0, 1]
callwait	5000	0	0 2605 1 2395 Name: callwait, dtype: int64	[1, 0]
forward	5000	0	0 2597 1 2403 Name: forward, dtype: int64	[1, 0]
confer	5000	0	0 2610 1 2390 Name: confer, dtype: int64	[1, 0]
ebill	5000	0	0 3257 1 1743 Name: ebill, dtype: int64	[0, 1]
owntv	5000	0	1 4915 0 85 Name: owntv, dtype: int64	[1, 0]
ownvcr	5000	0	1 4578 0 422 Name: ownvcr, dtype: int64	[1, 0]
owndvd	5000	0	1 4568 0 432 Name: owndvd, dtype: int64	[1, 0]
owncd	5000	0	1 4664 0 336 Name: owncd, dtype: int64	[0, 1]
ownpda	5000	0	0 3995 1 1005 Name: ownpda, dtype: int64	[0, 1]
ownpc	5000	0	1 3164 0 1836 Name: ownpc, dtype: int64	[0, 1]
ownipod	5000	0	0 2604 1 2396 Name: ownipod, dtype: int64	[1, 0]
owngame	5000	0	0 2626 1 2374 Name: owngame, dtype: int64	[1, 0]
ownfax	5000	0	0 4106 1 894 Name: ownfax, dtype: int64	[0, 1]
news	5000	0	0 2637 1 2363 Name: news, dtype: int64	[0, 1]
response_01	5000	0	0 4582 1 418 Name: response_01, dtype: ...	[0, 1]
response_02	5000	0	0 4351 1 649 Name: response_02, dtype: ...	[1, 0]
response_03	5000	0	0 4487 1 513 Name: response_03, dtype: ...	[0, 1]

84 rows × 4 columns

Handling Outliers

There are some extreame high or extream low value which need to rectify so that it will not have impact on our model

In [18]:
```
#Handling Outliers for numerical data - Through function

def outlier_capping(x):
    x = x.clip_upper(x.quantile(0.99))
```

```
    x = x.clip_lower(x.quantile(0.01))
    return x

custdata_df_num = custdata_df_num.apply(lambda x: outlier_cappin
g(x))
```

Handling Missing Values

As we can see in the above audit report some data is missing in nmerical data set and some are missing in categorical.

We will treat numerical missing data with mean() and categorical data with mode()

In [19]:
```
#Handling missings - by Function (Makethe function to treat all
data in one shot)

def Missing_imputation(x):
    x = x.fillna(x.mean())
    return x

custdata_df_num = custdata_df_num.apply(lambda x: Missing_imputa
tion(x))
```

In [20]:
```
#Handling missings - by Function (Makethe function to treat all
data in one shot)

def Missing_imputation(x):
    x = x.fillna(x.mode())
    return x

custdata_df_cat = custdata_df_cat.apply(lambda x: Missing_imputa
tion(x))
```

Again check missing values has been treated or not

In [21]:
```
# Find the total number of missing values in the numerical data
i.e custdata_df_num
print ("\nMissing values in numerical data :  ", custdata_df_nu
m.isnull().sum().values.sum())
print ("\nMissing values in categorical data :  ", custdata_df_n
um.isnull().sum().values.sum())
```

```
Missing values in numerical data :   0

Missing values in categorical data :   0
```

Dummy creation for categorical data

In [22]:
```
# We need to create a function for dummy creation
def create_dummies(df, colname):
    col_dummies = pd.get_dummies(df[colname], prefix = colname)
    col_dummies.drop(col_dummies.columns[0], axis = 1, inplace =
True)
```

```
    df = pd.concat([df, col_dummies], axis = 1)
    df.drop(colname, axis = 1, inplace = True)
    return df
```

In [23]:
```
custdata_df_cat=custdata_df.select_dtypes(include=['object'])
cat_varlist=list(custdata_df_cat.columns)
```

In [24]:
```
# For customer_features in categorical features

for c_feature in cat_varlist:
    custdata_df_cat[c_feature]=custdata_df_cat[c_feature].astype
('category')
    custdata_df_cat=create_dummies(custdata_df_cat,c_feature)
```

In [25]:
```
custdata_df_cat.sample(5)
```

Out[25]:

	region_2	region_3	region_4	region_5	townsize_2.0	townsize_3.0	towns
1760	1	0	0	0	1	0	
789	0	0	1	0	0	1	
1830	0	0	1	0	0	0	
2970	0	0	0	0	0	0	
4839	0	0	0	0	0	0	

Merge Numerical and categorical data

Now we have nice simple clean data for numerical and categorical both data frame, so now we can merge both data set

In [26]:
```
custdata_df_new = pd.concat([custdata_df_num, custdata_df_cat],
axis=1)
```

In [27]:
```
custdata_df_new.head()
```

Out[27]:

	age	ed	income	lninc	debtinc	creddebt	lncreddebt	othdebt	lnothc
0	20.0	15.0	31.0	3.433987	11.1	1.200909	0.183079	2.240091	0.806
1	22.0	17.0	15.0	2.708050	18.6	1.222020	0.200505	1.567980	0.449
2	67.0	14.0	35.0	3.555348	9.9	0.928620	-0.074056	2.536380	0.930
3	23.0	16.0	20.0	2.995732	5.7	0.033160	-3.401690	1.117200	0.110
4	26.0	16.0	23.0	3.135494	1.7	0.214659	-1.538705	0.176341	-1.735

In [27]:
```
# Some features has highly correlation with their log valuesso I
am going to drop it.

#custdata_df_new.drop(['cardten', 'lninc','lncreddebt','lnothdeb
t','lnlongmon','lnlongten','lntollmon','lntollten',
#'lnequipmon','lnequipten','lncardten','lnwiremon','lnwireten'],
```

```
axis=1, inplace=True)
```

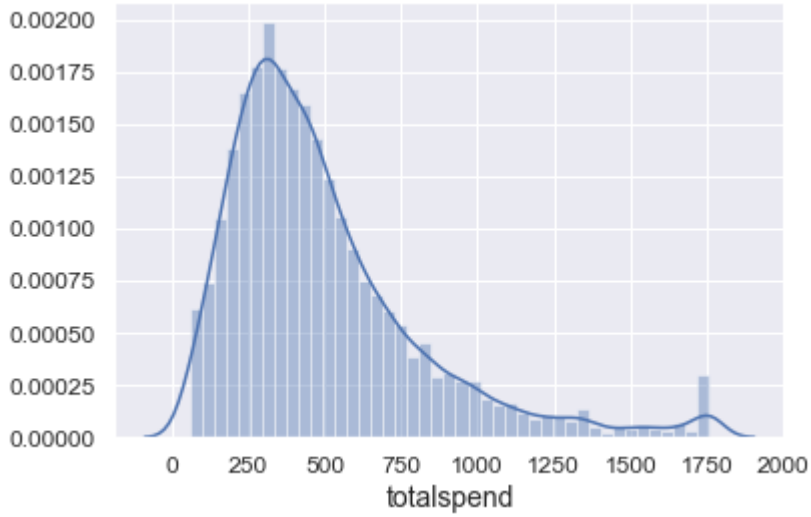
In [28]:
```
custdata_df_new.shape
```

Out[28]:
```
(5000, 400)
```

Explore data and check the variable distribution

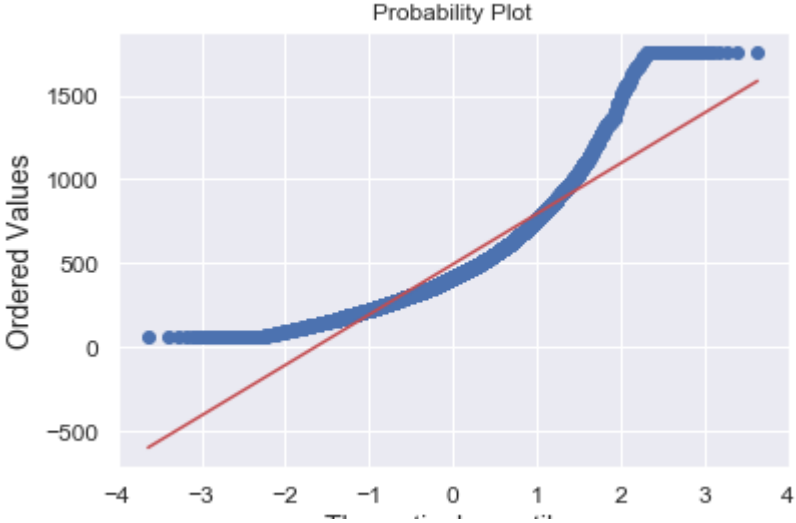
In [29]:
```
# For Linear regressin y(totalspend) should follow normal istrib
ution
import seaborn as sns
sns.distplot(custdata_df_new.totalspend)
```

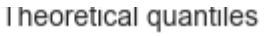
Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc5866a0>
```



Now as you can see distribution is right skewed so we need to take log and then plot the graph.

In [30]:
```
from scipy import stats
import pylab

stats.probplot(custdata_df_new.totalspend, dist="norm", plot=pyl
ab )
pylab.show()
```



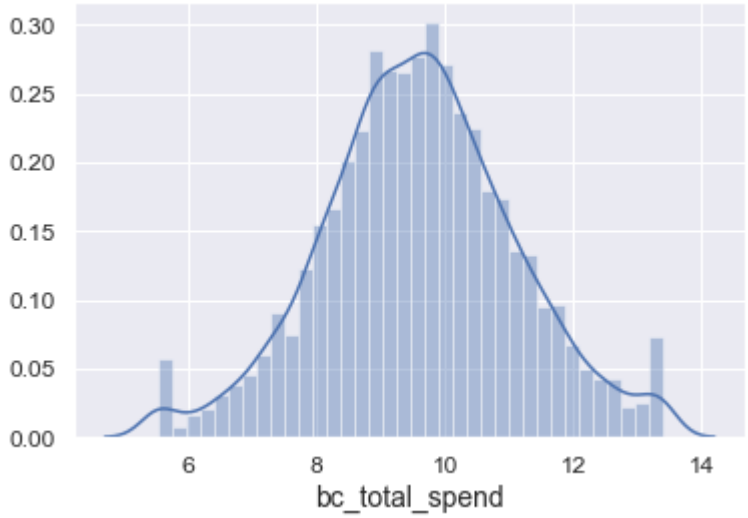


As we can see above graph our data is not distributed normally perfectly, so we will use boxcox techniue to make it perfect normal

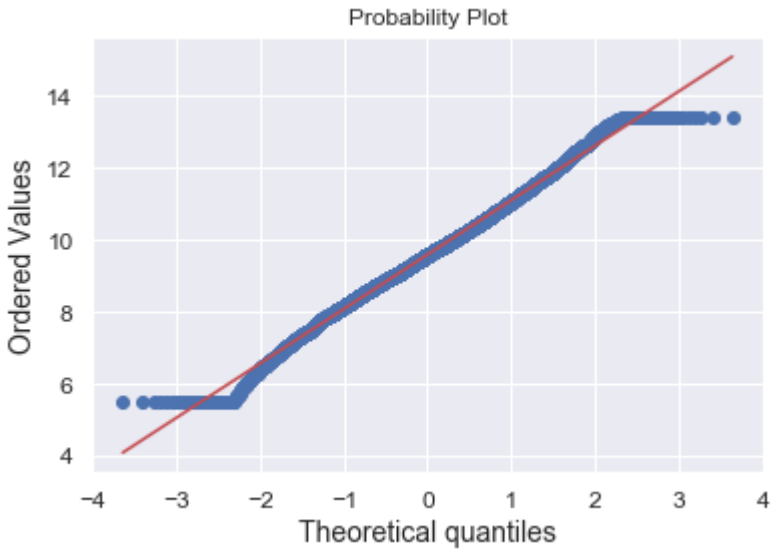
In [31]:
```
from scipy import stats

# transform training data & save lambda value
custdata_df_new['bc_total_spend'],fitted_lambda = stats.boxcox(c
ustdata_df_new['totalspend'])
```

In [32]:
```
sns.distplot(custdata_df_new.bc_total_spend)
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc010390>
```



In [33]:
```
stats.probplot(custdata_df_new.bc_total_spend, dist="norm", plot
=pylab )
pylab.show()
```



In [34]:
```
#Drop the y variables(totalspend) from dataframe because we are
looking correlation between all x variables

custdata_df_new.drop(['totalspend'],axis=1,inplace=True)
```

Devide data into train and test data

In [35]:
```
#Splitting the data in all x variable and y variable.

feature_columns=custdata_df_new.columns.difference(['bc_total_spend'])
```

In [36]:
```
from sklearn.model_selection import train_test_split

train_x,test_x,train_y,test_y=train_test_split(custdata_df_new[feature_columns],
                                               custdata_df_new['bc_total_spend'],
                                               test_size=0.2,
                                               random_state=12)
```

In [37]:
```
print (len(train_x))
print (len(test_x))
print (len(train_y))
print (len(test_y))
```

```
4000
1000
4000
1000
```

Feature selection by random forrest

In [38]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

In [39]:
```
RandomForestRegressor?
```

In [40]:
```
param_grid={'n_estimators':np.arange(10,25)}

tree=GridSearchCV(RandomForestRegressor(oob_score=False,warm_start=True),param_grid,cv=2)
tree.fit(train_x,train_y)
```

Out[40]:
```
GridSearchCV(cv=2, error_score='raise',
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=True),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_estimators': array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [41]:
```
tree.best_params_
```

Out[41]:
```
{'n_estimators': 23}
```

In [42]:
```
# we can take n_estimators': 23
radm_clf = RandomForestRegressor(oob_score=True,n_estimators=23)
radm_clf.fit( train_x, train_y)
```

Out[42]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_dept
h=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=23, n_job
s=1,
           oob_score=True, random_state=None, verbose=0, warm_s
tart=False)
```

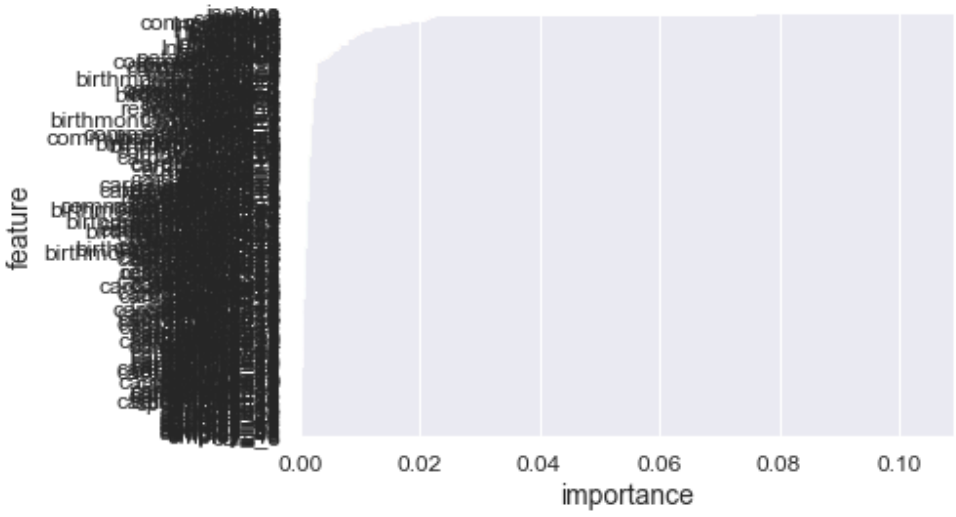
In [43]:
```
radm_clf.oob_score_
```

Out[43]:
```
0.15767032576065787
```

In [44]:
```
from sklearn import metrics
```

In [45]:
```
indices = np.argsort(radm_clf.feature_importances_)[::-1]
feature_rank = pd.DataFrame( columns = ['rank', 'feature', 'impo
rtance'] )
for f in range(train_x.shape[1]):
  feature_rank.loc[f] = [f+1,
                         train_x.columns[indices[f]],
                         radm_clf.feature_importances_[indices
[f]]]
sns.barplot( y = 'feature', x = 'importance', data = feature_ran
k )
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc09e4a8>
```



- As above graph is not clear so we will use below method.

In [46]:
```
indices = np.argsort(radm_clf.feature_importances_)[::-1]
```

```
feature_rank = pd.DataFrame( columns = ['rank', 'feature', 'impo
rtance'] )
for f in range(train_x.shape[1]):
  feature_rank.loc[f] = [f+1,
                         train_x.columns[indices[f]],
                         radm_clf.feature_importances_[indices
[f]]]

feature_rank
```

Out[46]:

	rank	feature	importance
0	1	income	1.037454e-01
1	2	lninc	7.589780e-02
2	3	card_3	2.596244e-02
3	4	card_2	2.239525e-02
4	5	carvalue	2.154065e-02
5	6	card_4	2.112358e-02
6	7	reason_2	2.094437e-02
7	8	debtinc	1.930640e-02
8	9	commutetime	1.846777e-02
9	10	hourstv	1.661541e-02
10	11	age	1.652895e-02
11	12	lncreddebt	1.468481e-02
12	13	creddebt	1.400680e-02
13	14	othdebt	1.242060e-02
14	15	ed	1.229365e-02
15	16	lncardmon	1.194613e-02
16	17	tenure	1.115788e-02
17	18	lnothdebt	1.040137e-02
18	19	pets	1.001150e-02
19	20	lncardten	9.964655e-03
20	21	cardmon	9.230549e-03
21	22	lntollmon	9.129050e-03
22	23	longmon	9.106174e-03
23	24	card_5	9.043518e-03
24	25	spoused	8.085497e-03
25	26	cardten	7.930730e-03
26	27	lnlongten	7.926722e-03
27	28	lnequipten	7.834682e-03
28	29	lnlongmon	7.799020e-03
29	30	longten	7.240294e-03
...	...	...	...
369	370	employ_45	8.859233e-05
370	371	address_45	7.747346e-05

371	372	owntv_1	6.995432e-05
372	373	reason_3	5.521577e-05
373	374	address_53	5.338786e-05
374	375	inccat_5	5.133915e-05
375	376	employ_38	5.067010e-05
376	377	address_33	4.766309e-05
377	378	address_23	4.622708e-05
378	379	employ_26	3.523784e-05
379	380	cars_7	3.163522e-05
380	381	address_46	3.095265e-05
381	382	address_41	1.586377e-05
382	383	employ_36	1.417776e-05
383	384	address_43	4.679721e-06
384	385	address_49	2.893917e-06
385	386	address_50	2.708603e-06
386	387	employ_42	7.787210e-07
387	388	employ_47	1.211027e-07
388	389	address_55	1.107699e-07
389	390	address_52	4.552084e-08
390	391	address_54	9.325467e-09
391	392	employ_52	0.000000e+00
392	393	address_57	0.000000e+00
393	394	employ_51	0.000000e+00
394	395	employ_41	0.000000e+00
395	396	employ_49	0.000000e+00
396	397	employ_48	0.000000e+00
397	398	employ_46	0.000000e+00
398	399	cars_8	0.000000e+00

399 rows × 3 columns

In [47]:
```
#Select features and then convert it into list
x=feature_rank.loc[0:75,['feature']]
x=x['feature'].tolist()
print(x)
```

```
['income', 'lninc', 'card_3', 'card_2', 'carvalue', 'card_4',
'reason_2', 'debtinc', 'commutetime', 'hourstv', 'age', 'lncred
debt', 'creddebt', 'othdebt', 'ed', 'lncardmon', 'tenure', 'lno
thdebt', 'pets', 'lncardten', 'cardmon', 'lntollmon', 'longmo
n', 'card_5', 'spoused', 'cardten', 'lnlongten', 'lnequipten',
'lnlongmon', 'longten', 'card2_2', 'lntollten', 'lnequipmon',
'tollmon', 'card2_3', 'lnwiremon', 'gender_1', 'reside', 'tollt
en', 'pets_cats', 'pets_dogs', 'pets_freshfish', 'equipmon', 'c
ommutewalk_1', 'polparty_1', 'lnwireten', 'polview_4', 'card2be
nefit_3', 'union_1', 'spousedcat_2', 'card2benefit_2', 'card2ty
```

```
pe_2', 'vote_1', 'carown_0', 'equipten', 'region_3', 'card2_4',
'carbought_1', 'carbuy_1', 'jobsat_4', 'birthmonth_October', 'a
ctive_1', 'townsize_3.0', 'cardbenefit_4', 'region_5', 'cartype
_0', 'cardtype_2', 'cardtype_4', 'forward_1', 'commutebus_1',
'jobsat_3', 'commutecar_1', 'card2benefit_4', 'townsize_4.0',
'commuterail_1', 'wiremon']
```

In [48]:
```
# Create data frame with selected features
rf_features=['income', 'lninc', 'card_3', 'card_2', 'carvalue',
'card_4', 'commutetime', 'reason_2', 'debtinc', 'hourstv', 'cred
debt', 'age', 'lncreddebt', 'tenure', 'lncardmon', 'lnothdebt',
'ed', 'othdebt', 'pets', 'lncardten', 'lntollmon', 'cardmon', 'l
ongmon', 'card_5', 'lnlongmon', 'reside', 'spoused', 'cardten',
'lntollten', 'lnequipmon', 'lnlongten', 'tollten', 'longten', 't
ollmon', 'lnwiremon', 'card2_2', 'card2_3', 'pets_freshfish', 'l
nequipten', 'gender_1', 'pets_dogs', 'equipten', 'wireten', 'pet
s_cats', 'address_22', 'equipmon', 'wiremon', 'jobsat_4', 'commu
tebus_1', 'card2benefit_4']

rf_features.append('bc_total_spend')
df_rf= custdata_df_new[rf_features]

df_rf.head(5)
```

Out[48]:

	income	lninc	card_3	card_2	carvalue	card_4	commutetime	reason_2
0	31.0	3.433987	1	0	14.3	0	22.0	0
1	15.0	2.708050	0	1	6.8	0	29.0	0
2	35.0	3.555348	0	1	18.8	0	24.0	1
3	20.0	2.995732	0	1	8.7	0	38.0	0
4	23.0	3.135494	0	0	10.6	1	32.0	0

Check - 2: Find multicolinearlity

In [49]:
```
import statsmodels as sm
from statsmodels.stats.outliers_influence import variance_inflat
ion_factor
from patsy import dmatrices
```

In [50]:
```
del rf_features[-1]
```

In [51]:
```
%%capture
#gather features
features = "+".join(rf_features)
```

In [52]:
```
features
```

Out[52]:
```
'income+lninc+card_3+card_2+carvalue+card_4+commutetime+reason_
2+debtinc+hourstv+creddebt+age+lncreddebt+tenure+lncardmon+lnot
hdebt+ed+othdebt+pets+lncardten+lntollmon+cardmon+longmon+card_
5+lnlongmon+reside+spoused+cardten+lntollten+lnequipmon+lnlongt
en+tollten+longten+tollmon+lnwiremon+card2_2+card2_3+pets_fresh
fish+lnequipten+gender_1+pets_dogs+equipten+wireten+pets_cats+a
```

```
address_22+equipmon+wiremon+jobsat_4+commutebus_1+card2benefit_
4'
```

In [53]:
```
# get y and X dataframes based on this regression:
y, X = dmatrices('bc_total_spend~' + features,df_rf, return_type
='dataframe')
```

In [54]:
```
# For each X, calculate VIF and save in dataframe
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for
i in range(X.shape[1])]
vif["features"] = X.columns
```

In [55]:
```
vif.sort_values(by=['VIF Factor'],ascending=False)
```

Out[55]:

	VIF Factor	features
0	1686.859676	Intercept
23	148.928795	longmon
33	117.386519	longten
25	33.762250	lnlongmon
31	31.415359	lnlongten
14	24.098226	tenure
32	21.097455	tollten
42	19.936232	equipten
34	15.002808	tollmon
46	14.944802	equipmon
28	14.474511	cardten
19	13.324262	pets
47	13.134309	wiremon
43	12.799070	wireten
38	11.869724	pets_freshfish
22	10.962786	cardmon
2	9.948275	lninc
1	9.725833	income
29	8.130066	lntollten
20	7.905704	lncardten
16	7.517424	lnothdebt
39	6.498882	lnequipten
9	6.486941	debtinc
15	5.797448	lncardmon
18	5.167748	othdebt
13	4.276376	lncreddebt
11	3.947735	creddebt
5	3.534663	carvalue
21	3.065122	lntollmon

30	2.381248	lnequipmon
26	2.006284	reside
12	1.972903	age
4	1.882678	card_2
3	1.856482	card_3
27	1.833635	spoused
44	1.833349	pets_cats
6	1.788595	card_4
41	1.742336	pets_dogs
35	1.599433	lnwiremon
17	1.465111	ed
37	1.311558	card2_3
36	1.294873	card2_2
24	1.257129	card_5
48	1.039194	jobsat_4
10	1.037751	hourstv
8	1.022514	reason_2
40	1.021022	gender_1
45	1.019974	address_22
7	1.015482	commutetime
49	1.013662	commutebus_1
50	1.006757	card2benefit_4

In [56]:
```
#Select only those features whose Vif is greater than 10.
vif1=vif[vif['VIF Factor']>10].reset_index().loc[:,['features']]
vif1.drop([0],axis=0,inplace=True)
drop_vars= vif1["features"].tolist()
drop_vars
```

Out[56]:
```
['tenure',
 'pets',
 'cardmon',
 'longmon',
 'lnlongmon',
 'cardten',
 'lnlongten',
 'tollten',
 'longten',
 'tollmon',
 'pets_freshfish',
 'equipten',
 'wireten',
 'equipmon',
 'wiremon']
```

In [57]:
```
# dropping variables that have VIF greater than 10
df_rf.drop(drop_vars,axis=1,inplace=True)
```

In [58]:
```
df_rf.shape
```

Out[58]: (5000, 36)

Check 3 - All x variables should have a linear relationship with Y

In [59]:
```
sns.lmplot(x="income",y="bc_total_spend",data=df_rf,aspect=5,sca
tter_kws={'alpha':0.5})
```

Out[59]: <seaborn.axisgrid.FacetGrid at 0xb319320>

Splitting data for training and testing

In [60]:
```
# Dropping variables one at a time which have p-values greater t
han 5%
feature_columns=df_rf.columns.difference(['bc_total_spend','addr
ess_22','pets_cats','lncardmon','lnequipten','spoused'
                                          ,'creddebt','reside','o
thdebt','debtinc','lnothdebt','jobsat_4','income','carvalue',
                                          'card2benefit_4','commu
tebus_1','hourstv','lnwiremon','pets_dogs','lncardten',
                                          'commutetime','lntollte
n','lntollmon','lnequipmon'])
```

In [61]:
```
train_x,test_x,train_y,test_y=train_test_split(df_rf[feature_col
umns],
                                               df_rf['bc_total_sp
end'],
                                                test_size=.2,
                                                random_state=123)
```

Build Regression model using statsmodels.api

In [62]:
```
import statsmodels.api as sm
```

In [63]:
```
train_x = sm.add_constant(train_x)
test_x=sm.add_constant(test_x)
lm=sm.OLS(train_y,train_x).fit()
```

In [64]:
```
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================
===============
Dep. Variable:         bc_total_spend   R-squared:
0.325
Model:                            OLS   Adj. R-squared:
```

```
0.323
Method:                 Least Squares   F-statistic:
159.7
Date:                Tue, 01 Oct 2019   Prob (F-statistic):
0.00
Time:                        22:46:22   Log-Likelihood:
-6598.7
No. Observations:                4000   AIC:
1.322e+04
Df Residuals:                    3987   BIC:
1.331e+04
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.
025      0.975]
------------------------------------------------------------------------------
const          8.2401      0.172     47.920      0.000       7.
903       8.577
age           -0.0042      0.001     -3.681      0.000      -0.
006      -0.002
card2_2       -0.3794      0.052     -7.349      0.000      -0.
481      -0.278
card2_3       -0.3452      0.051     -6.788      0.000      -0.
445      -0.246
card_2        -1.2433      0.064    -19.525      0.000      -1.
368      -1.118
card_3        -1.2391      0.064    -19.465      0.000      -1.
364      -1.114
card_4        -1.2693      0.060    -21.022      0.000      -1.
388      -1.151
card_5        -0.9928      0.108     -9.214      0.000      -1.
204      -0.782
ed            -0.0154      0.007     -2.349      0.019      -0.
028      -0.003
gender_1      -0.1489      0.040     -3.713      0.000      -0.
227      -0.070
lncreddebt     0.0396      0.020      1.997      0.046       0.
001       0.079
lninc          0.8090      0.034     23.600      0.000       0.
742       0.876
reason_2       0.7071      0.081      8.746      0.000       0.
549       0.866
==============================================================================
Omnibus:                        4.553   Durbin-Watson:
1.976
Prob(Omnibus):                  0.103   Jarque-Bera (JB):
4.573
Skew:                          -0.082   Prob(JB):
0.102
Kurtosis:                       2.983   Cond. No.
473.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the er
rors is correctly specified.
```

In [65]:
```
lm.pvalues.sort_values(ascending=False)
```

Out[65]:
```
lncreddebt     4.587652e-02
ed             1.887734e-02
age            2.357317e-04
gender_1       2.078595e-04
card2_3        1.303811e-11
card2_2        2.408523e-13
reason_2       3.210677e-18
card_5         4.947423e-20
card_3         1.090371e-80
card_2         3.707595e-81
card_4         3.890897e-93
lninc         2.235272e-115
const          0.000000e+00
dtype: float64
```

In [66]:
```
print('Parameters:', lm.params) # Find the parameters of x i.e B
eta value
print('R2: ', lm.rsquared) # Find the r**2
```

```
Parameters: const         8.240115
age           -0.004194
card2_2       -0.379415
card2_3       -0.345223
card_2        -1.243274
card_3        -1.239082
card_4        -1.269314
card_5        -0.992801
ed            -0.015444
gender_1      -0.148872
lncreddebt     0.039624
lninc          0.809044
reason_2       0.707096
dtype: float64
R2:  0.3246950341215388
```

Eveluation of model accuracy

In [67]:
```
test_pred=lm.predict(test_x)
train_pred=lm.predict(train_x)

from sklearn import metrics

print('MSE Test:',metrics.mean_squared_error(test_y,test_pred))
print('MSE Train:',metrics.mean_squared_error(train_y,train_pre
d))
```

```
MSE Test: 1.5275403610908898
MSE Train: 1.5864409028377782
```

In [68]:
```
print ('MAE:', metrics.mean_absolute_error(test_y, test_pred))
print ('MSE:', metrics.mean_squared_error(test_y, test_pred))
print ('RMSE:', np.sqrt(metrics.mean_squared_error(test_y, test_
pred)))
```

```
MAE: 0.9762061585444958
MSE: 1.5275403610908898
```

```
RMSE: 1.2359370376725871
```

In [69]:
```
MAPE_train = '%.3f' % np.mean(np.abs(train_y-train_pred)/(train_
y))
MAPE_test = '%.3f' % np.mean(np.abs(test_y-test_pred)/(test_y))

# print the values of MAPE for train and test
print('MAPE of training data: ', MAPE_train,  ' | ', 'MAPE of te
sting data: ', MAPE_test)
```

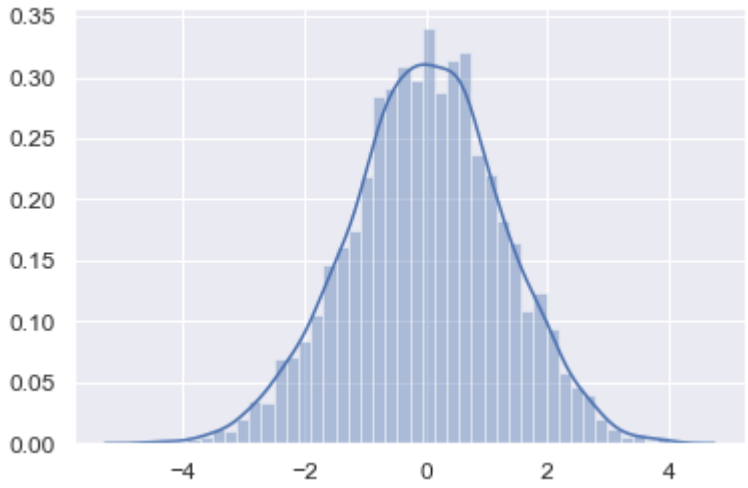
```
MAPE of training data:  0.110  |  MAPE of testing data:  0.106
```

Check Normality and Residuals

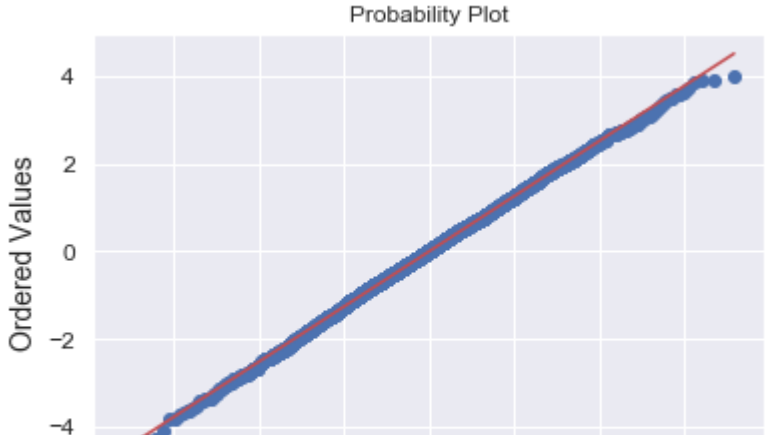
We will use Q-Q plot to examine this

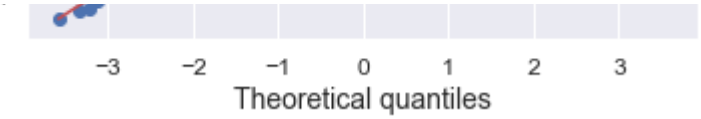
In [70]:
```
residuals=train_y-train_pred

import seaborn as sns

sns.distplot(residuals)
```

Out[70]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xf889278>
```



In [71]:
```
from scipy import stats
import pylab

stats.probplot(residuals,dist='norm',plot=pylab)
pylab.show()
```





From the above histogram and Q-Q plot, shows that the residuals are normally distributed, so our assumption is not violated

Nov 10, 2019 (http://www.datasciencelovers.com/2019/11/)

Store Sales Prediction – Forecasting (http://www.datasciencelovers.com/machine-learning-projects/store-sales-prediction-forecasting/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/store-sales-prediction-forecasting/) Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/)** Tag **linear regression (http://www.datasciencelovers.com/tag/linear-regression/), machine learning (http://www.datasciencelovers.com/tag/machine-learning/), store sales predection (http://www.datasciencelovers.com/tag/store-sales-predection/), supervised learning (http://www.datasciencelovers.com/tag/supervised-learning/)**

Business Context:

The objective is predicting store sales using historical markdown data. One challenge of modelling retail data is the need to make decisions based on limited history. If Christmas comes but once a year, so does the chance to see how strategic decisions impacted the bottom line.

Business Problem:

Company provided with historical sales data for 45 Walmart stores located in different regions. Each store contains a number of departments, and you are tasked with predicting the department-wide sales for each store.

In addition, Walmart runs several promotional markdown events throughout the year. These markdowns precede prominent holidays, the four largest of which are the Super Bowl, Labour Day, Thanksgiving, and Christmas. The weeks including these holidays are weighted five times higher in the evaluation than non-holiday weeks. Part of the challenge presented by this competition is modelling the effects of markdowns on these holiday weeks in the absence of complete/ideal historical data.

Data Availability:

stores.csv: This file contains anonymized information about the 45 stores, indicating the type and size of store.

train.csv: This is the historical training data, which covers to 2010-02-05 to 2012-11- 01, Within this file you will find the following fields:

- **Store –** the store number
- **Dept –** the department number
- **Date –** the week
- **Weekly_Sales –** sales for the given department in the given store
- **IsHoliday –** whether the week is a special holiday week

test.csv: This file is identical to train.csv, except we have withheld the weekly sales. You must predict the sales for each triplet of store, department, and date in this file.

features.csv: This file contains additional data related to the store, department, and regional activity for the given dates. It contains the following fields:

- **Store –** the store number
- **Date –** the week
- **Temperature –** average temperature in the region
- **Fuel_Price –** cost of fuel in the region
- **MarkDown1-5 –** anonymized data related to promotional markdowns that Walmart is running. MarkDown data is only available after Nov 2011, and is not available for all stores all the time. Any missing value is marked with an NA.
- **CPI –** the consumer price index
- **Unemployment –** the unemployment rate
- **IsHoliday –** whether the week is a special holiday week

Let’s develop a machine learning model for further analysis.

Store Sales Prediction With Linear Regression.

Import Library

In [1]:
```
#Basic python library which need to import
import pandas as pd
import numpy as np

#Date stuff
from datetime import datetime
from datetime import timedelta

#Library for Nice graphing
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as sn
%matplotlib inline

#Library for statistics operation
import scipy.stats as stats

# Date Time library
from datetime import datetime

#Machine learning Library
import statsmodels.api as sm
from sklearn import metrics
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error, mean_squared_er
ror

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Settings
pd.set_option('display.max_columns', None)
np.set_printoptions(threshold=np.nan)
np.set_printoptions(precision=3)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_valida
tion.py:41: DeprecationWarning: This module was deprecated in v
ersion 0.18 in favor of the model_selection module into which a
ll the refactored classes and functions are moved. Also note th
at the interface of the new CV iterators are different from tha
```

```
at the interface of the new CV iterators are different from tha
t of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\wei
ght_boosting.py:29: DeprecationWarning: numpy.core.umath_tests
is an internal NumPy module and should not be imported. It will
be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

Import Data from Csv files

In [2]:
```
# Importing training data set
train = pd.read_csv("train.csv")

#Import Test Data
test=pd.read_csv("test.csv")

# Import Store data set
stores = pd.read_csv("stores.csv")

# Now import features data set
feature = pd.read_csv("features.csv")
```

Merge the data sets:

-(train+Store+Feature)
-(test+Store+Feature)

In [3]:
```
# For Train data set
train_bt = pd.merge(train,stores)
train = pd.merge(train_bt,feature)

#For test data set
test_bt = pd.merge(test,stores)
test= pd.merge(test_bt,feature)
```

In [4]:
```
train.head(2)
```

Out[4]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Type	Size	Temperature	Fu
0	1	1	2010-02-05	24924.50	False	A	151315	42.31	
1	1	2	2010-02-05	50605.27	False	A	151315	42.31	

In [5]:
```
test.head(2)
```

Out[5]:

	Store	Dept	Date	IsHoliday	Type	Size	Temperature	Fuel_Price	MarkD
0	1	1	2012-11-02	False	A	151315	55.32	3.386	67
1	1	2	2012-	False	A	151315	55.32	3.386	67

11-02

In [6]:
```
print (train.info())
print ("****************************************")
print (test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 421570 entries, 0 to 421569
Data columns (total 16 columns):
Store           421570 non-null int64
Dept            421570 non-null int64
Date            421570 non-null object
Weekly_Sales    421570 non-null float64
IsHoliday       421570 non-null bool
Type            421570 non-null object
Size            421570 non-null int64
Temperature     421570 non-null float64
Fuel_Price      421570 non-null float64
MarkDown1       150681 non-null float64
MarkDown2       111248 non-null float64
MarkDown3       137091 non-null float64
MarkDown4       134967 non-null float64
MarkDown5       151432 non-null float64
CPI             421570 non-null float64
Unemployment    421570 non-null float64
dtypes: bool(1), float64(10), int64(3), object(2)
memory usage: 51.9+ MB
None
****************************************
<class 'pandas.core.frame.DataFrame'>
Int64Index: 115064 entries, 0 to 115063
Data columns (total 15 columns):
Store           115064 non-null int64
Dept            115064 non-null int64
Date            115064 non-null object
IsHoliday       115064 non-null bool
Type            115064 non-null object
Size            115064 non-null int64
Temperature     115064 non-null float64
Fuel_Price      115064 non-null float64
MarkDown1       114915 non-null float64
MarkDown2       86437 non-null float64
MarkDown3       105235 non-null float64
MarkDown4       102176 non-null float64
MarkDown5       115064 non-null float64
CPI             76902 non-null float64
Unemployment    76902 non-null float64
dtypes: bool(1), float64(9), int64(3), object(2)
memory usage: 13.3+ MB
None
```

Select only positive weekly sales

In [7]:
```
# tale only those values whose sales is positive.
train = train[train['Weekly_Sales']>0]
```

Data Description:

1. Training Data

In [8]:
```
numeric_var_train=[key for key in dict(train.dtypes) if dict(tra
in.dtypes)[key] in ['float64', 'int64', 'float32', 'int32']]
cat_var_train=[key for key in dict(train.dtypes) if dict(train.d
types)[key] in ['object']]

# Train Numerical Data
train_num=train[numeric_var_train]

# Train Categorical Data
train_cat=train[cat_var_train]

print (numeric_var_train)
print (cat_var_train)
```

```
['Store', 'Dept', 'Weekly_Sales', 'Size', 'Temperature', 'Fuel_
Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'Ma
rkDown5', 'CPI', 'Unemployment']
['Date', 'Type']
```

In [12]:
```
# Use a general function that returns multiple values
def var_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.sum(), x.me
an(), x.median(),  x.std(), x.var(), x.min(), x.dropna().quantil
e(0.01), x.dropna().quantile(0.05),x.dropna().quantile(0.10),x.d
ropna().quantile(0.25),x.dropna().quantile(0.50),x.dropna().quan
tile(0.75), x.dropna().quantile(0.90),x.dropna().quantile(0.95),
x.dropna().quantile(0.99),x.max()],
                  index=['N', 'NMISS', 'SUM', 'MEAN','MEDIAN',
'STD', 'VAR', 'MIN', 'P1' , 'P5' ,'P10' ,'P25' ,'P50' ,'P75' ,'P
90' ,'P95' ,'P99' ,'MAX'])
```

In [ ]:
```
num_summary=train_num.apply(lambda x: var_summary(x)).T
num_summary
```

In [9]:
```
def cat_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.value_count
s()],
                  index=['N', 'NMISS', 'ColumnsNames'])

cat_summary=train_cat.apply(lambda x: cat_summary(x))
cat_summary
```

Out[9]:

	Date	Type
N	420212	420212
NMISS	0	0
ColumnsNames	2011-12-23 3018 2011-11-25 3016 2011-12-...	A 214961 B 162787 C 42464 Name: Type...

2. Testing Data

In [10]:
```
numeric_var_test=[key for key in dict(test.dtypes) if dict(test.
dtypes)[key] in ['float64', 'int64', 'float32', 'int32']]
cat_var_test=[key for key in dict(test.dtypes) if dict(test.dtyp
es)[key] in ['object']]

# Train Numerical Data
test_num=test[numeric_var_test]

# Train Categorical Data
test_cat=test[cat_var_test]

print (numeric_var_test)
print (cat_var_test)
```

```
['Store', 'Dept', 'Size', 'Temperature', 'Fuel_Price', 'MarkDow
n1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5', 'CPI',
'Unemployment']
['Date', 'Type']
```

In [13]:
```
# Numerical data summary report
num_summary=test_num.apply(lambda x: var_summary(x)).T

num_summary.head()
```

Out[13]:

	N	NMISS	SUM	MEAN	MEDIAN	
Store	115064.0	0.0	2.558817e+06	22.238207	22.000	12.
Dept	115064.0	0.0	5.101883e+06	44.339524	37.000	30.
Size	115064.0	0.0	1.570597e+10	136497.688921	140167.000	61106.
Temperature	115064.0	0.0	6.206760e+06	53.941804	54.470	18.
Fuel_Price	115064.0	0.0	4.121070e+05	3.581546	3.606	0.

In [19]:
```
# categorical data summary report
def cat_summary(x):
    return pd.Series([x.count(), x.isnull().sum(), x.value_count
s()],
                     index=['N', 'NMISS', 'ColumnsNames'])

cat_summary=test_cat.apply(lambda x: cat_summary(x))
cat_summary
```

Out[19]:

	Date	Type
N	115064	115064
NMISS	0	0
ColumnsNames	2012-12-21 3002 2012-12-07 2989 2012-12-...	A 58713 B 44500 C 11851 Name: Type, d...

In [21]:
```
# Run Pandas profilingto see the over all report
import pandas_profiling
pandas_profiling.ProfileReport(train)
```

Out[21]:

Overview

Dataset info

Number of variables	16
Number of observations	421570
Total Missing (%)	21.1%
Total size in memory	51.9 MiB
Average record size in memory	129.0 B

Variables types

Numeric	13
Categorical	2
Boolean	1
Date	0
Text (Unique)	0
Rejected	0
Unsupported	0

Warnings

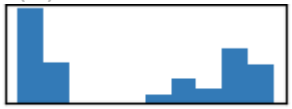
- `Date` has a high cardinality: 143 distinct values Warning
- `MarkDown1` has 270889 / 64.3% missing values Missing
- `MarkDown2` has 310322 / 73.6% missing values Missing
- `MarkDown3` has 284479 / 67.5% missing values Missing
- `MarkDown4` has 286603 / 68.0% missing values Missing
- `MarkDown5` has 270138 / 64.1% missing values Missing

Variables

CPI

Numeric

Distinct count	2145
Unique (%)	0.5%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	171.2
Minimum	126.06
Maximum	227.23
Zeros (%)	0.0%



Toggle details

Date

Categorical

Distinct count	143
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0

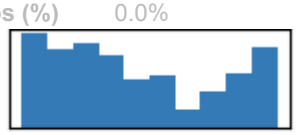
2011-12-23	3027
2011-11-25	3021
2011-12-16	3013
Other values (140)	412509

Toggle details

Dept

Numeric

Distinct count	81
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	44.26
Minimum	1
Maximum	99
Zeros (%)	0.0%

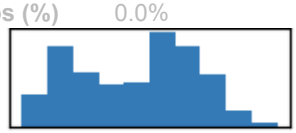


Toggle details

Fuel_Price

Numeric

Distinct count	892
Unique (%)	0.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.361
Minimum	2.472
Maximum	4.468
Zeros (%)	0.0%



Toggle details

IsHoliday

Boolean

Distinct count	2
Unique (%)	0.0%

Missing (%)	0.0%
Missing (n)	0
Mean	0.070358

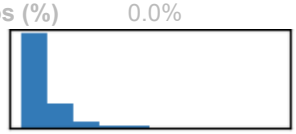
True	29661
(Missing)	391909

Toggle details

MarkDown1

Numeric

Distinct count	2278
Unique (%)	0.5%
Missing (%)	64.3%
Missing (n)	270889
Infinite (%)	0.0%
Infinite (n)	0
Mean	7246.4
Minimum	0.27
Maximum	88647
Zeros (%)	0.0%

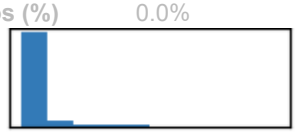


Toggle details

MarkDown2

Numeric

Distinct count	1500
Unique (%)	0.4%
Missing (%)	73.6%
Missing (n)	310322
Infinite (%)	0.0%
Infinite (n)	0
Mean	3334.6
Minimum	-265.76
Maximum	104520
Zeros (%)	0.0%



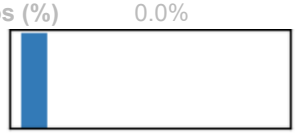
Toggle details

MarkDown3

Numeric

Distinct count	1663
Unique (%)	0.4%
Missing (%)	67.5%
Missing (n)	284479
Infinite (%)	0.0%
Infinite (n)	0
Mean	1439.4

Minimum	-29.1
Maximum	141630
Zeros (%)	0.0%

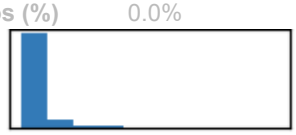


Toggle details

MarkDown4

Numeric

Distinct count	1945
Unique (%)	0.5%
Missing (%)	68.0%
Missing (n)	286603
Infinite (%)	0.0%
Infinite (n)	0
Mean	3383.2
Minimum	0.22
Maximum	67475
Zeros (%)	0.0%

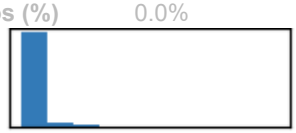


Toggle details

MarkDown5

Numeric

Distinct count	2294
Unique (%)	0.5%
Missing (%)	64.1%
Missing (n)	270138
Infinite (%)	0.0%
Infinite (n)	0
Mean	4629
Minimum	135.16
Maximum	108520
Zeros (%)	0.0%



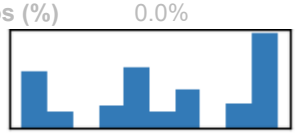
Toggle details

Size

Numeric

Distinct count	40
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	136730
Minimum	34875
Maximum	219622
Zeros (%)	0.0%

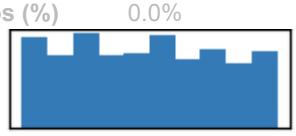


Toggle details

Store

Numeric

Distinct count	45
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	22.201
Minimum	1
Maximum	45
Zeros (%)	0.0%

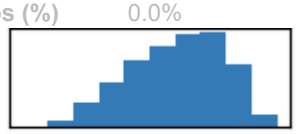


Toggle details

Temperature

Numeric

Distinct count	3528
Unique (%)	0.8%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	60.09
Minimum	-2.06
Maximum	100.14
Zeros (%)	0.0%



Toggle details

Type

Categorical

Distinct count	3
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0

A 215478

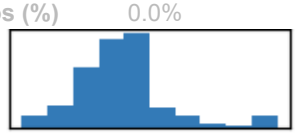
B	163495
C	42597

Toggle details

Unemployment

Numeric

Distinct count	349
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	7.9603
Minimum	3.879
Maximum	14.313
Zeros (%)	0.0%

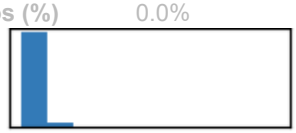


Toggle details

Weekly_Sales

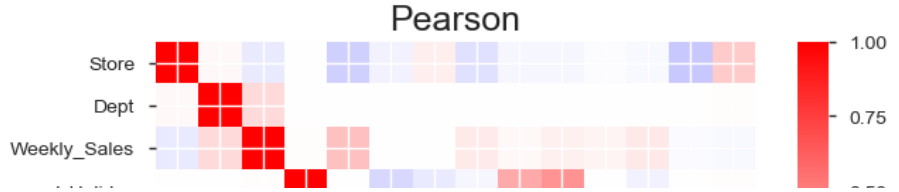
Numeric

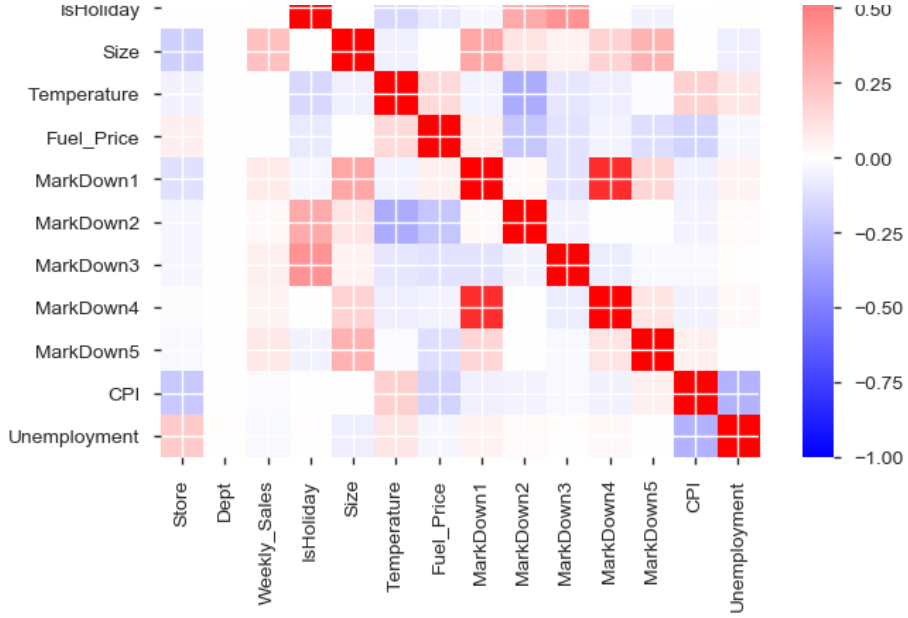
Distinct count	359464
Unique (%)	85.3%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	15981
Minimum	-4988.9
Maximum	693100
Zeros (%)	0.0%

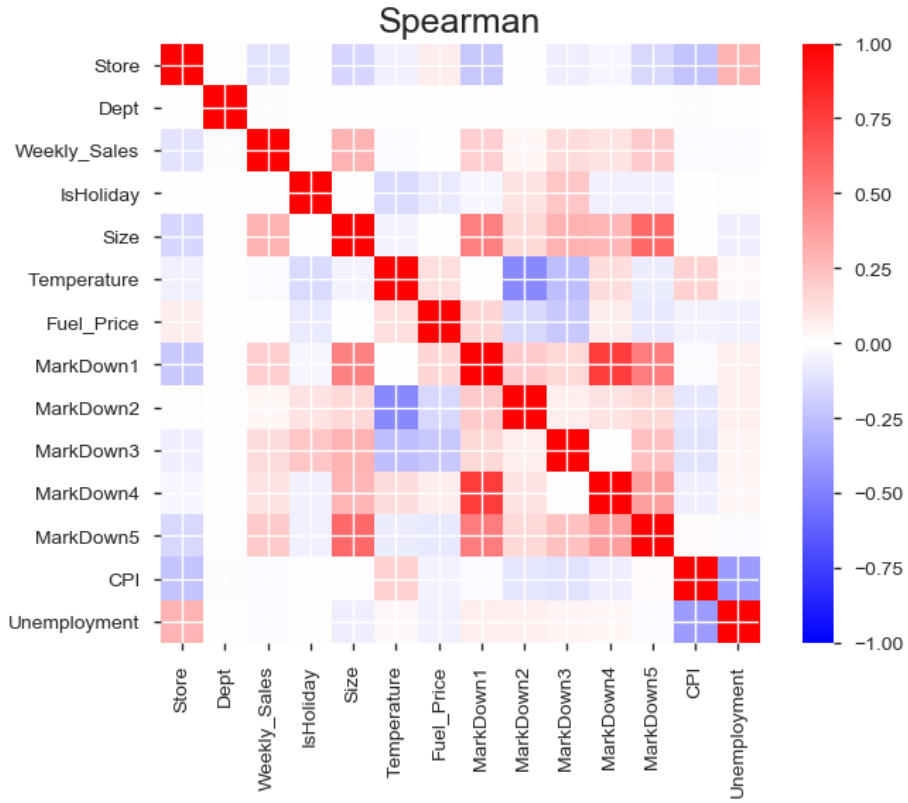


Toggle details

Correlations







Sample

	Store	Dept	Date	Weekly_Sales	IsHoliday	Type	Size	T
0	1	1	2010-02-05	24924.50	False	A	151315	
1	1	2	2010-02-05	50605.27	False	A	151315	
2	1	3	2010-02-05	13740.12	False	A	151315	
3	1	4	2010-02-05	39954.04	False	A	151315	
4	1	5	2010-02-05	32229.38	False	A	151315	

In [22]:
```
pandas_profiling.ProfileReport(test)
```

Out[22]:

Overview

Dataset info

Number of variables	15
Number of observations	115064
Total Missing (%)	7.4%
Total size in memory	13.3 MiB
Average record size in memory	121.0 B

Variables types

Numeric	12
Categorical	2
Boolean	1
Date	0
Text (Unique)	0
Rejected	0
Unsupported	0

Warnings

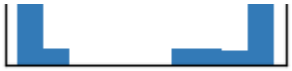
- CPI has 38162 / 33.2% missing values Missing
- MarkDown2 has 28627 / 24.9% missing values Missing
- MarkDown3 has 9829 / 8.5% missing values Missing
- MarkDown4 has 12888 / 11.2% missing values Missing
- MarkDown5 is highly skewed (γ1 = 37.977) Skewed
- Unemployment has 38162 / 33.2% missing values Missing

Variables

CPI

Numeric

Distinct count	361
Unique (%)	0.3%
Missing (%)	33.2%
Missing (n)	38162
Infinite (%)	0.0%
Infinite (n)	0
Mean	176.96
Minimum	131.24
Maximum	228.98
Zeros (%)	0.0%



Toggle details

Date

Categorical

Distinct count	39
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0

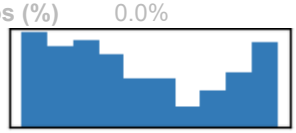
2012-12-21	3002
2012-12-07	2989
2012-12-28	2988
Other values (36)	106085

Toggle details

Dept

Numeric

Distinct count	81
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	44.34
Minimum	1
Maximum	99
Zeros (%)	0.0%



Toggle details

Fuel_Price

Numeric

Distinct count	297
Unique (%)	0.3%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.5815
Minimum	2.872
Maximum	4.125
Zeros (%)	0.0%

Toggle details

IsHoliday

Boolean

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Mean	0.077592

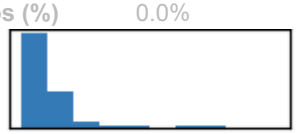
True	8928
(Missing)	106136

Toggle details

MarkDown1

Numeric

Distinct count	1753
Unique (%)	1.5%
Missing (%)	0.1%
Missing (n)	149
Infinite (%)	0.0%
Infinite (n)	0
Mean	7689.2
Minimum	-2781.4
Maximum	103180
Zeros (%)	0.0%

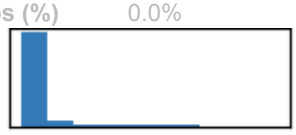


Toggle details

MarkDown2

Numeric

Distinct count	1258
Unique (%)	1.1%
Missing (%)	24.9%
Missing (n)	28627
Infinite (%)	0.0%
Infinite (n)	0
Mean	3734.1
Minimum	-35.74
Maximum	71074
Zeros (%)	0.0%

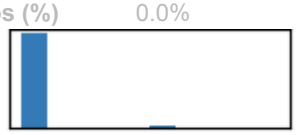


Toggle details

MarkDown3

Numeric

Distinct count	1422
Unique (%)	1.2%
Missing (%)	8.5%
Missing (n)	9829
Infinite (%)	0.0%
Infinite (n)	0
Mean	2403.1
Minimum	-179.26
Maximum	149480
Zeros (%)	0.0%

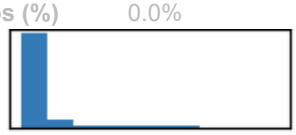


Toggle details

MarkDown4

Numeric

Distinct count	1484
Unique (%)	1.3%
Missing (%)	11.2%
Missing (n)	12888
Infinite (%)	0.0%
Infinite (n)	0
Mean	3356.2
Minimum	0.22
Maximum	65345
Zeros (%)	0.0%

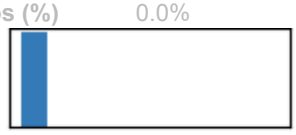


Toggle details

MarkDown5

Numeric

Distinct count	1754
Unique (%)	1.5%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3922.7
Minimum	-185.17
Maximum	771450
Zeros (%)	0.0%

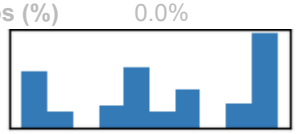


Toggle details

Size

Numeric

Distinct count	40
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	136500
Minimum	34875
Maximum	219622
Zeros (%)	0.0%

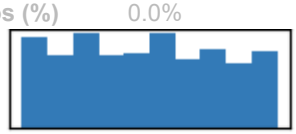


Toggle details

Store

Numeric

Distinct count	45
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	22.238
Minimum	1
Maximum	45
Zeros (%)	0.0%

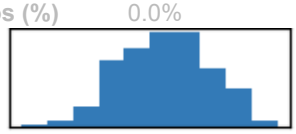


Toggle details

Temperature

Numeric

Distinct count	1236
Unique (%)	1.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	53.942
Minimum	-7.29
Maximum	101.95
Zeros (%)	0.0%



Toggle details

Type

Categorical

Distinct count	3
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0

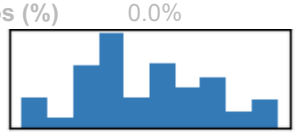
A	58713
B	44500
C	11851

Toggle details

Unemployment

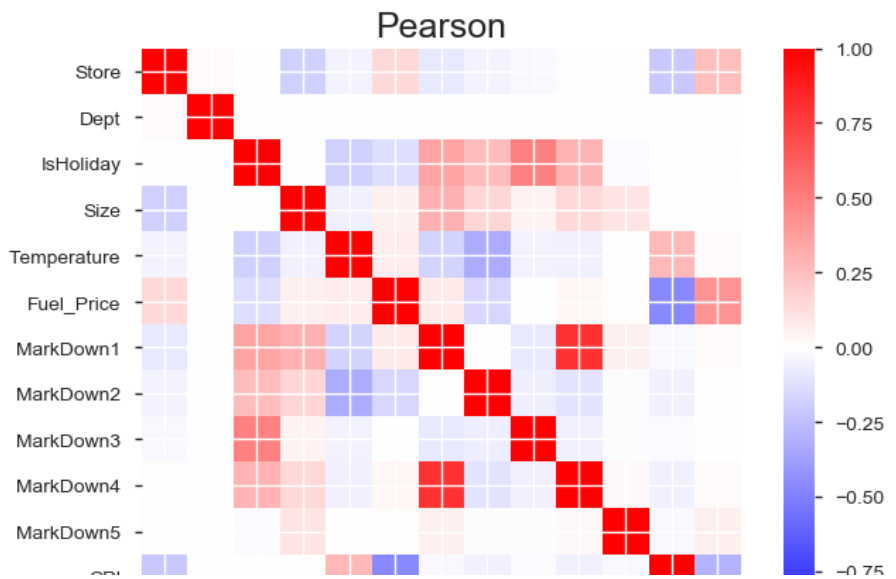
Numeric

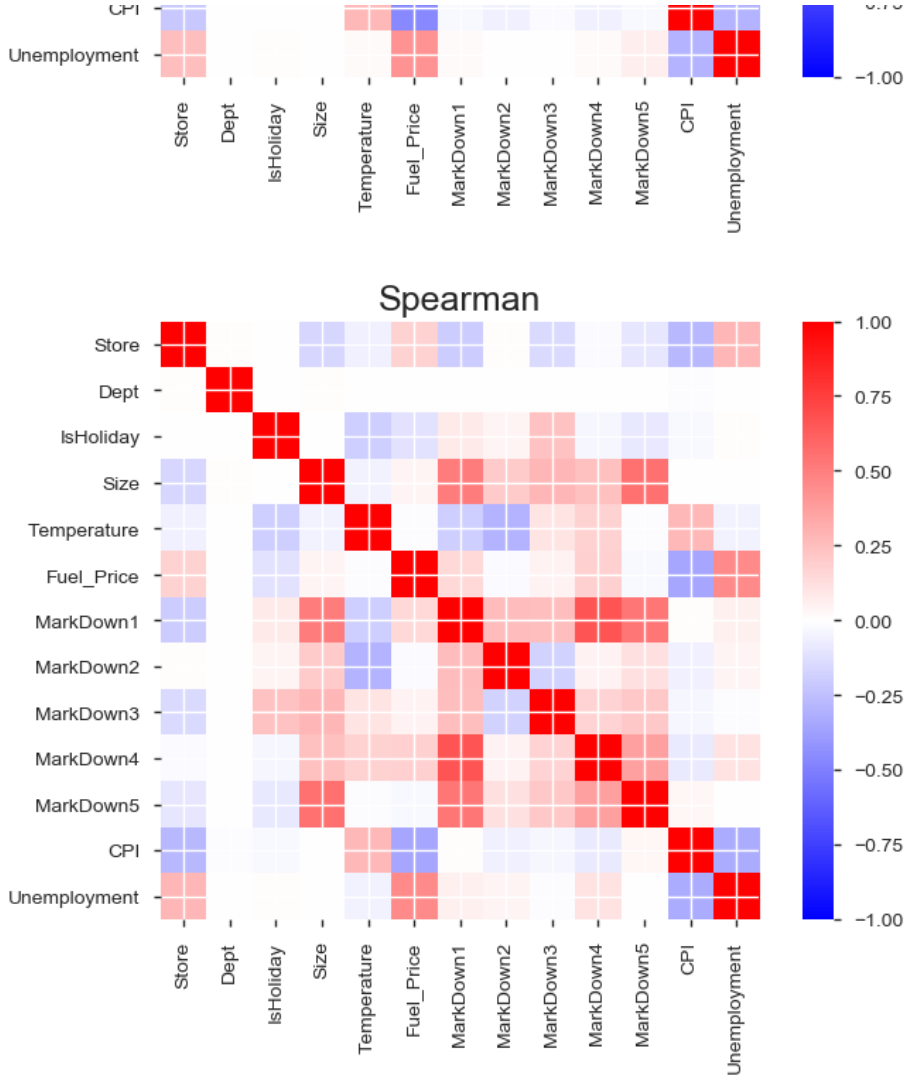
Distinct count	90
Unique (%)	0.1%
Missing (%)	33.2%
Missing (n)	38162
Infinite (%)	0.0%
Infinite (n)	0
Mean	6.8687
Minimum	3.684
Maximum	10.199
Zeros (%)	0.0%



Toggle details

Correlations





Sample

	Store	Dept	Date	IsHoliday	Type	Size	Temperature	Fu
0	1	1	2012-11-02	False	A	151315	55.32	
1	1	2	2012-11-02	False	A	151315	55.32	
2	1	3	2012-11-02	False	A	151315	55.32	
3	1	4	2012-11-02	False	A	151315	55.32	
4	1	5	2012-11-02	False	A	151315	55.32	

Correlation matrix

In [14]:
```
# Correlation for train data
train_corr=pd.DataFrame(train.corr())
```

```
train_corr.head()
```

Out[14]:

	Store	Dept	Weekly_Sales	IsHoliday	Size	Temper
Store	1.000000	0.024258	-0.085117	-0.000522	-0.182763	-0.05
Dept	0.024258	1.000000	0.148749	0.000663	-0.002491	0.00
Weekly_Sales	-0.085117	0.148749	1.000000	0.012843	0.244117	-0.00
IsHoliday	-0.000522	0.000663	0.012843	1.000000	0.000797	-0.15
Size	-0.182763	-0.002491	0.244117	0.000797	1.000000	-0.05

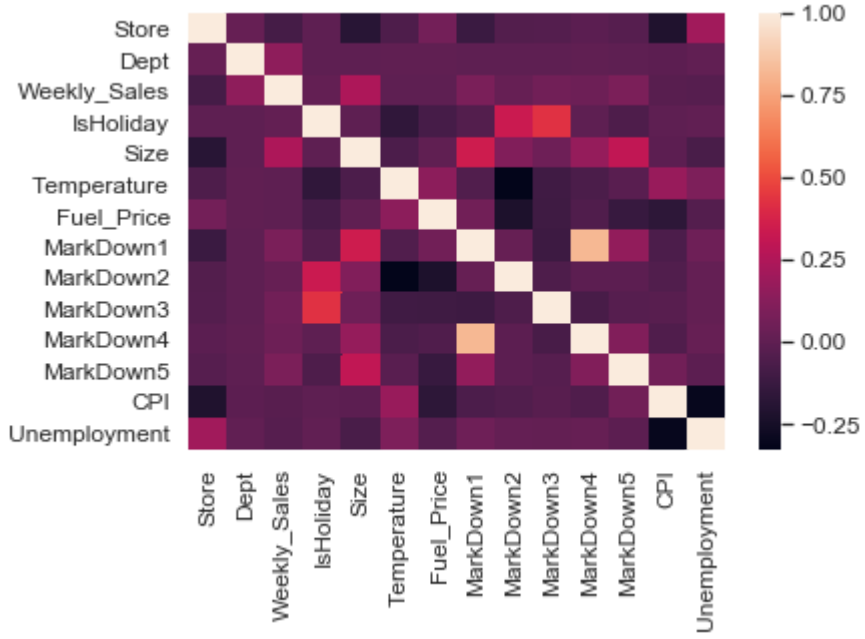
In [15]:
```
# Correlation for test data
test_corr=pd.DataFrame(test.corr())
test_corr.head()
```

Out[15]:

	Store	Dept	IsHoliday	Size	Temperature	Fuel_Price
Store	1.000000	0.019627	-0.001166	-0.186845	-0.043495	0.153425
Dept	0.019627	1.000000	0.001249	0.001502	0.003970	0.000554
IsHoliday	-0.001166	0.001249	1.000000	-0.000443	-0.187428	-0.126443
Size	-0.186845	0.001502	-0.000443	1.000000	-0.061256	0.055088
Temperature	-0.043495	0.003970	-0.187428	-0.061256	1.000000	0.073938

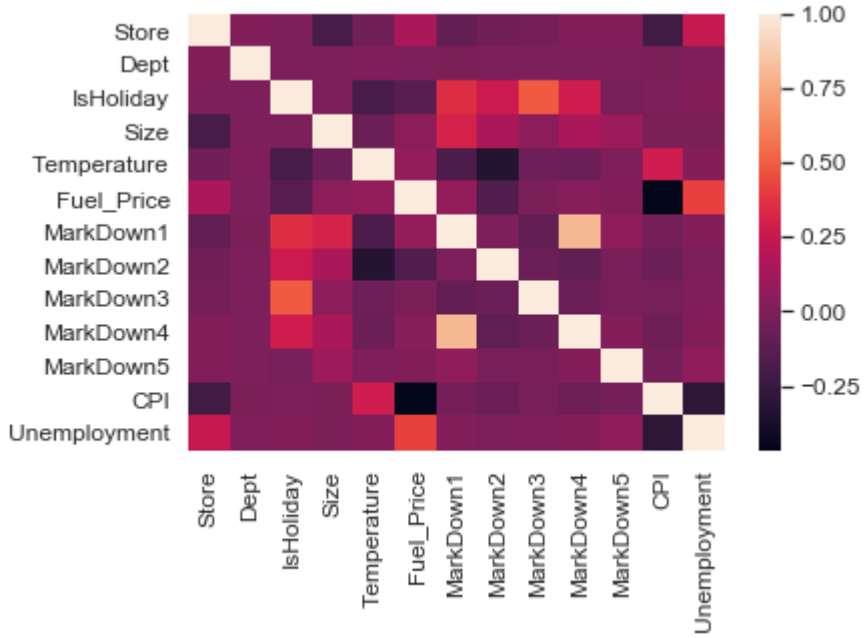
In [16]:
```
# visualize correlation matrix in Seaborn using a heatmap
sns.heatmap(train.corr())
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0xdb38b38>



In [17]:
```
# visualize correlation matrix in Seaborn using a heatmap
sns.heatmap(test.corr())
```

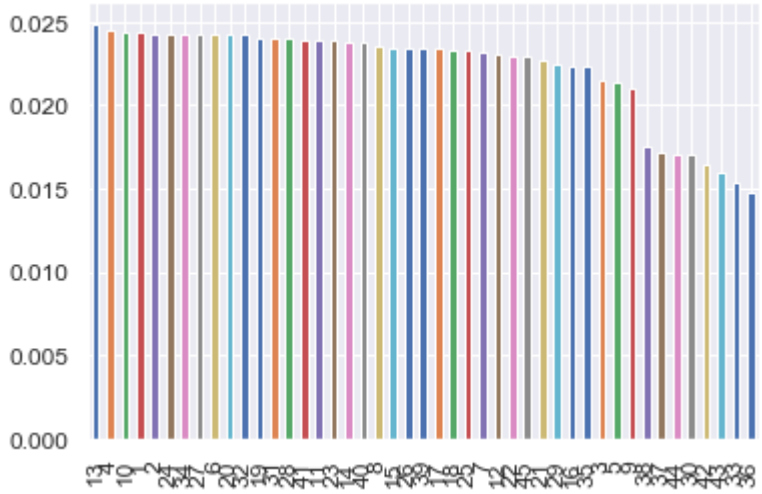
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1870d4e0>



Data Exploratory Analysis:

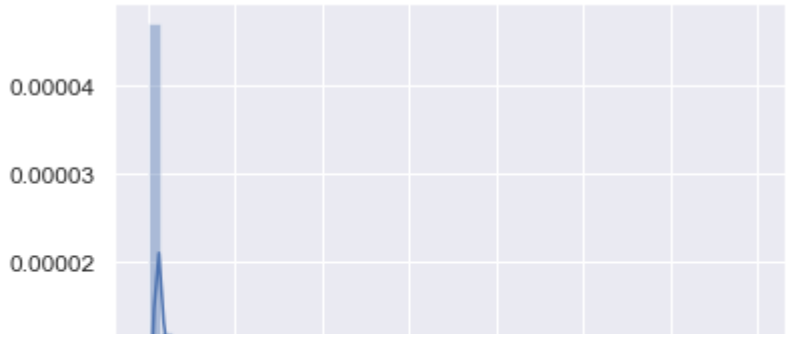
In [18]:
```
# Store wise sales plot
train['Store'].value_counts(normalize=True).plot(kind = 'bar',fig=(4,5))
```

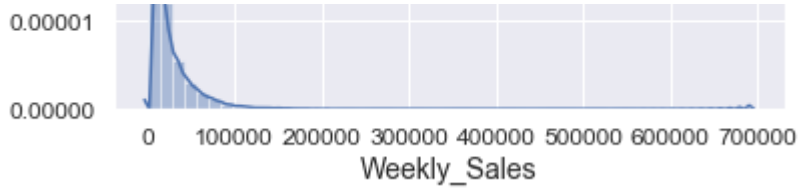
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x19c1e400>



In [19]:
```
# weekly sales plot
sns.distplot(train.Weekly_Sales)
```

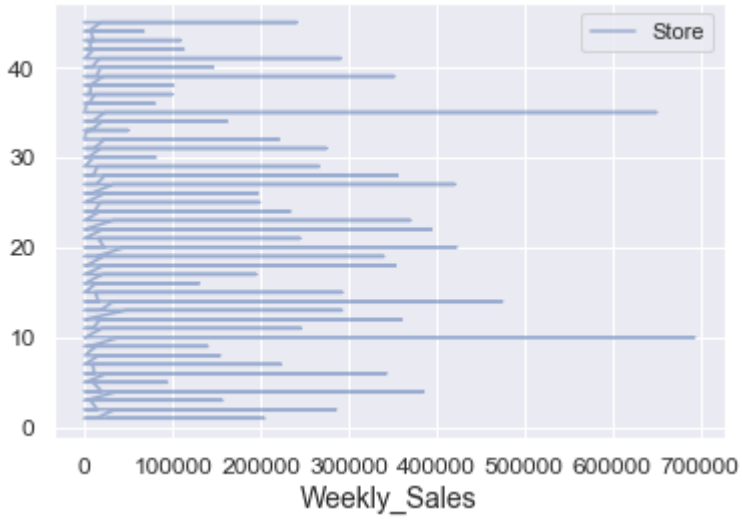
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x19c1e1d0>





In [20]:
```
# Store wise sales
train.plot(kind='line', x='Weekly_Sales', y='Store', alpha=0.5)
```

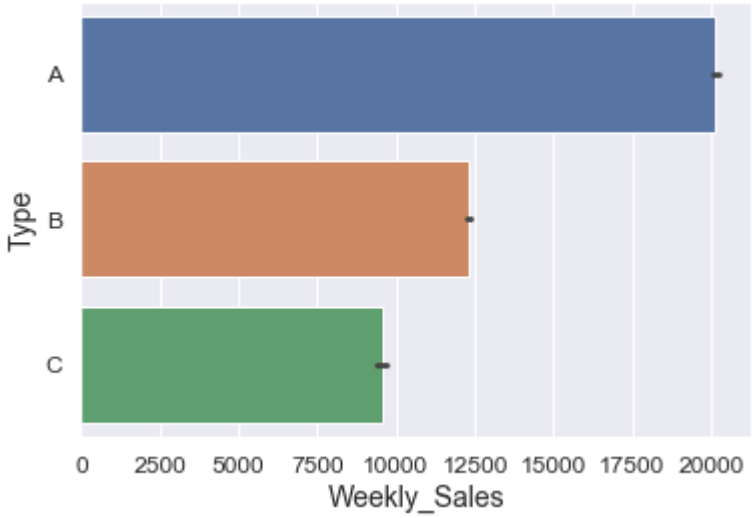
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x573d198>



Sales Vs Type:

In [21]:
```
# Weekly sales Type wise
sns.barplot(x=train["Weekly_Sales"],y=train["Type"])
```

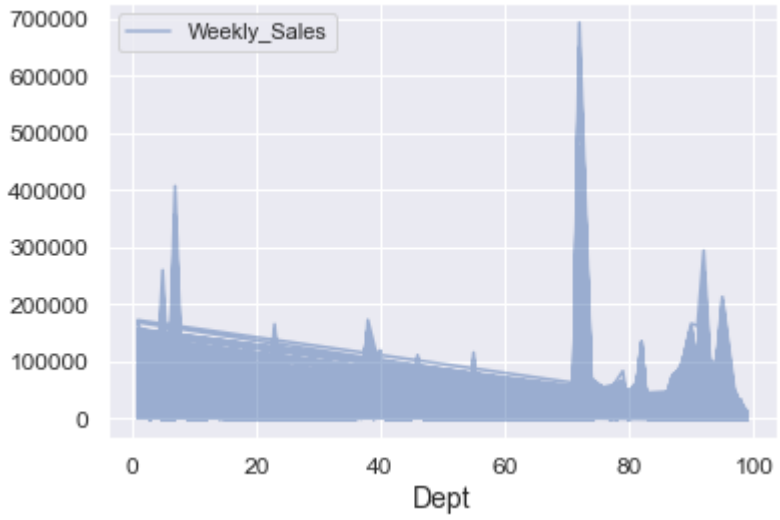
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1249be80>



Department Weise sales:

In [22]:
```
train.plot(kind='line', x='Dept', y='Weekly_Sales', alpha=0.5,fi
g=(4,5))
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x175ffd68>



Missing value Treatment

In [23]:
```
print (train.isnull().sum())
print ("*"*30)
print (test.isnull().sum())
```

```
Store                0
Dept                 0
Date                 0
Weekly_Sales         0
IsHoliday            0
Type                 0
Size                 0
Temperature          0
Fuel_Price           0
MarkDown1       270031
MarkDown2       309308
MarkDown3       283561
MarkDown4       285694
MarkDown5       269283
CPI                  0
Unemployment         0
dtype: int64
******************************
Store                0
Dept                 0
Date                 0
IsHoliday            0
Type                 0
Size                 0
Temperature          0
Fuel_Price           0
MarkDown1          149
MarkDown2        28627
MarkDown3         9829
MarkDown4        12888
MarkDown5            0
CPI              38162
Unemployment     38162
dtype: int64
```

Imputing it with its mean

In [24]:
```
test['CPI']=test.groupby(['Dept'])['CPI'].transform(lambda x: x.
fillna(x.mean()))
test['Unemployment']=test.groupby(['Dept'])['Unemployment'].tran
sform(lambda x: x.fillna(x.mean()))
```

Other Missing Value Treatment like Markdown, Imputing it with Zero(No Markdown)

In [25]:
```
train=train.fillna(0)
test=test.fillna(0)
```

In [26]:
```
# Recheck the missing values.

print (train.isnull().sum())
print ("*"*30)
print (test.isnull().sum())
```

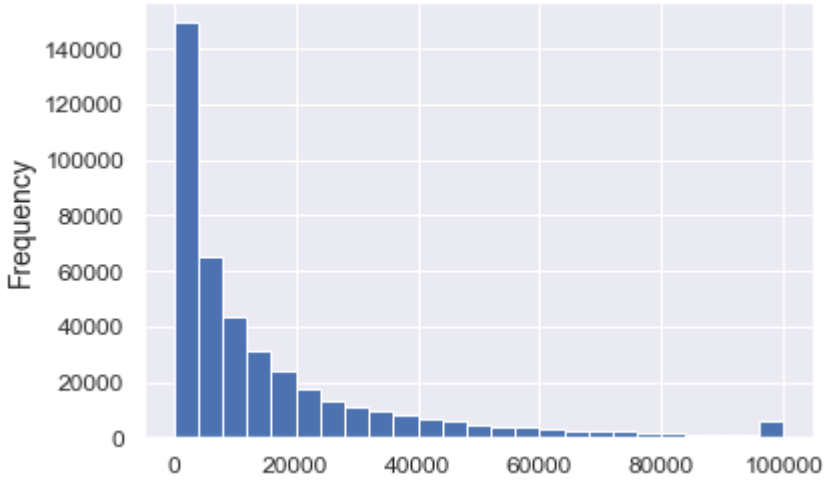
```
Store           0
Dept            0
Date            0
Weekly_Sales    0
IsHoliday       0
Type            0
Size            0
Temperature     0
Fuel_Price      0
MarkDown1       0
MarkDown2       0
MarkDown3       0
MarkDown4       0
MarkDown5       0
CPI             0
Unemployment    0
dtype: int64
******************************
Store           0
Dept            0
Date            0
IsHoliday       0
Type            0
Size            0
Temperature     0
Fuel_Price      0
MarkDown1       0
MarkDown2       0
MarkDown3       0
MarkDown4       0
MarkDown5       0
CPI             0
Unemployment    0
dtype: int64
```

Outlier Treatment

In [27]:
```
train.Weekly_Sales=np.where(train.Weekly_Sales>100000, 100000,train.Weekly_Sales)
```

In [28]:
```
train.Weekly_Sales.plot.hist(bins=25)
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x176834e0>
```



Feature Extraction

In this section, we select the appropriate features to train our classifier. Here, we create new features based on existing features. We also convert categorical features into numeric form.

In [29]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 420212 entries, 0 to 421569
Data columns (total 16 columns):
Store           420212 non-null int64
Dept            420212 non-null int64
Date            420212 non-null object
Weekly_Sales    420212 non-null float64
IsHoliday       420212 non-null bool
Type            420212 non-null object
Size            420212 non-null int64
Temperature     420212 non-null float64
Fuel_Price      420212 non-null float64
MarkDown1       420212 non-null float64
MarkDown2       420212 non-null float64
MarkDown3       420212 non-null float64
MarkDown4       420212 non-null float64
MarkDown5       420212 non-null float64
CPI             420212 non-null float64
Unemployment    420212 non-null float64
dtypes: bool(1), float64(10), int64(3), object(2)
memory usage: 71.7+ MB
```

Date Feature

In [30]:
```
train['Date'] = pd.to_datetime(train['Date'])
test['Date'] = pd.to_datetime(test['Date'])
```

In [31]:
```
# Extract date features
train['Date_dayofweek'] =train['Date'].dt.dayofweek
train['Date_month'] =train['Date'].dt.month
train['Date_year'] =train['Date'].dt.year
train['Date_day'] =train['Date'].dt.day

# For test data
test['Date_dayofweek'] =test['Date'].dt.dayofweek
test['Date_month'] =test['Date'].dt.month
test['Date_year'] =test['Date'].dt.year
test['Date_day'] =test['Date'].dt.day
```

Type Feature Details

In [32]:
```
print (train.Type.value_counts())
print ("*"*30)
print (test.Type.value_counts())
```

```
A    214961
B    162787
C     42464
Name: Type, dtype: int64
******************************
A    58713
B    44500
C    11851
Name: Type, dtype: int64
```

IsHoliday Feature Details

In [33]:
```
print (train.IsHoliday.value_counts())
print ("*"*30)
print (test.IsHoliday.value_counts())
```

```
False    390652
True      29560
Name: IsHoliday, dtype: int64
******************************
False    106136
True       8928
Name: IsHoliday, dtype: int64
```

In [34]:
```
# Combine train and test with Key
train_test_data = [train, test]
```

Converting Categorical Variable 'Type' into Numerical Variable For A=1 , B=2, C=3

In [35]:
```
type_mapping = {"A": 1, "B": 2, "C": 3}
for dataset in train_test_data:
    dataset['Type'] = dataset['Type'].map(type_mapping)
```

Converting Categorical Variable 'IsHoliday' into Numerical Variable

In [36]:
```
type_mapping = {False: 0, True: 1}
for dataset in train_test_data:
    dataset['IsHoliday'] = dataset['IsHoliday'].map(type_mappin
g)
```

Creating Extra Holiday Variable.If that week comes under extra holiday then 1(=Yes) else 2(=No)

Making New Holiday Variable Based on Given Data....

In [37]:
```
# For Train Data Set
train['Super_Bowl'] = np.where((train['Date']==datetime(2010, 2,
12)) | (train['Date']==datetime(2011, 2, 11)) | (train['Date']==
datetime(2012, 2, 10)) | (train['Date']==datetime(2013, 2, 8)),
1,0)
train['Labour_Day'] = np.where((train['Date']==datetime(2010, 9,
10)) | (train['Date']==datetime(2011, 9, 9)) | (train['Date']==d
atetime(2012, 9, 7)) | (train['Date']==datetime(2013, 9, 6)),1,
0)
train['Thanksgiving'] = np.where((train['Date']==datetime(2010,
11, 26)) | (train['Date']==datetime(2011, 11, 25)) | (train['Dat
e']==datetime(2012, 11, 23)) | (train['Date']==datetime(2013, 1
1, 29)),1,0)
train['Christmas'] = np.where((train['Date']==datetime(2010, 12,
31)) | (train['Date']==datetime(2011, 12, 30)) | (train['Date']=
=datetime(2012, 12, 28)) | (train['Date']==datetime(2013, 12, 2
7)),1,0)

#For Test Data se
t.....................................................................
.........
test['Super_Bowl'] = np.where((test['Date']==datetime(2010, 2, 1
2)) | (test['Date']==datetime(2011, 2, 11)) | (test['Date']==dat
etime(2012, 2, 10)) | (test['Date']==datetime(2013, 2, 8)),1,0)
test['Labour_Day'] = np.where((test['Date']==datetime(2010, 9, 1
0)) | (test['Date']==datetime(2011, 9, 9)) | (test['Date']==date
time(2012, 9, 7)) | (test['Date']==datetime(2013, 9, 6)),1,0)
test['Thanksgiving'] = np.where((test['Date']==datetime(2010, 1
1, 26)) | (test['Date']==datetime(2011, 11, 25)) | (test['Date']
==datetime(2012, 11, 23)) | (test['Date']==datetime(2013, 11, 2
9)),1,0)
test['Christmas'] = np.where((test['Date']==datetime(2010, 12, 3
1)) | (test['Date']==datetime(2011, 12, 30)) | (test['Date']==da
tetime(2012, 12, 28)) | (test['Date']==datetime(2013, 12, 27)),
1,0)
```

In [38]:
```
# Change the isHoliday value depending on these new holidays...
train['IsHoliday']=train['IsHoliday']|train['Super_Bowl']|train
['Labour_Day']|train['Thanksgiving']|train['Christmas']
test['IsHoliday']=test['IsHoliday']|test['Super_Bowl']|test['Lab
our_Day']|test['Thanksgiving']|test['Christmas']
```

In [39]:
```
# Count of holiday for train data
print (train.Christmas.value_counts())
print (train.Super_Bowl.value_counts())
```

```
print (train.Thanksgiving.value_counts())
print (train.Labour_Day.value_counts())
```

```
0    414303
1      5909
Name: Christmas, dtype: int64
0    411339
1      8873
Name: Super_Bowl, dtype: int64
0    414266
1      5946
Name: Thanksgiving, dtype: int64
0    411380
1      8832
Name: Labour_Day, dtype: int64
```

In [40]:
```
# Count of holiday for Test data
print (test.Christmas.value_counts())
print (test.Super_Bowl.value_counts())
print (test.Thanksgiving.value_counts())
print (test.Labour_Day.value_counts())
```

```
0    112076
1      2988
Name: Christmas, dtype: int64
0    112100
1      2964
Name: Super_Bowl, dtype: int64
0    112088
1      2976
Name: Thanksgiving, dtype: int64
0    115064
Name: Labour_Day, dtype: int64
```

In [41]:
```
# Since we have Imputed IsHoliday according to Extra holidays..T
hese extra holiday variable has redundant..
# Droping the Extra holiday variables because its redundant..
dp=['Super_Bowl','Labour_Day','Thanksgiving','Christmas']

train.drop(dp,axis=1,inplace=True)
test.drop(dp,axis=1,inplace=True)
```

In [42]:
```
train.head(2)
```

Out[42]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Type	Size	Temperature	Fu
0	1	1	2010-02-05	24924.50	0	1	151315	42.31	
1	1	2	2010-02-05	50605.27	0	1	151315	42.31	

Feature Selection

Droping irrevelent variable:

-Since we have imputed markdown variables therefore we will not be removing the all markdown variables.
-Removing MarkDown5 because its Highly Skewed.

In [43]:
```
features_drop=['Unemployment','CPI','MarkDown5']
train=train.drop(features_drop, axis=1)
test=test.drop(features_drop, axis=1)
```

In [44]:
```
train.head(2)
```

Out[44]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Type	Size	Temperature	Fu
0	1	1	2010-02-05	24924.50	0	1	151315	42.31	
1	1	2	2010-02-05	50605.27	0	1	151315	42.31	

In [45]:
```
test.head(2)
```

Out[45]:

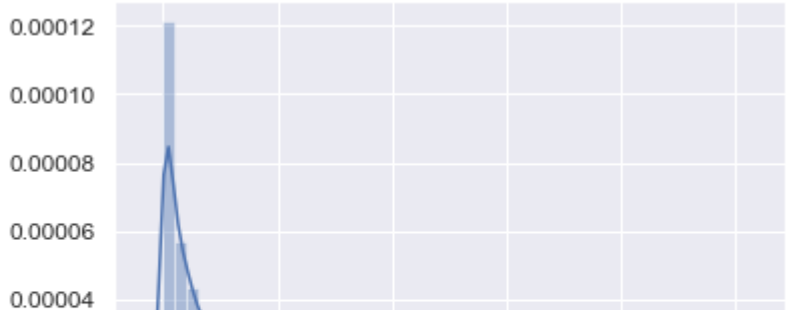
	Store	Dept	Date	IsHoliday	Type	Size	Temperature	Fuel_Price	MarkD
0	1	1	2012-11-02	0	1	151315	55.32	3.386	67
1	1	2	2012-11-02	0	1	151315	55.32	3.386	67

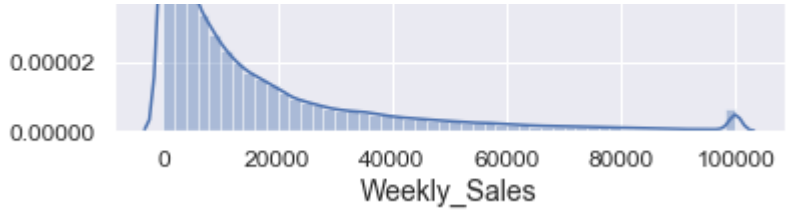
In [46]:
```
# Converting all float var to int integer..
for var in train:
    if train[var].dtypes == float:
        train[var]=train[var].astype(int)

for var in test:
    if test[var].dtypes == float:
        test[var]=test[var].astype(int)
```

First we should Check Y is normally distributed or not

In [47]:
```
import seaborn as sns
sns.distplot(train.Weekly_Sales)
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0xc2e0f98>





As we can see above fig y is not normally distributed so we will take log of Y

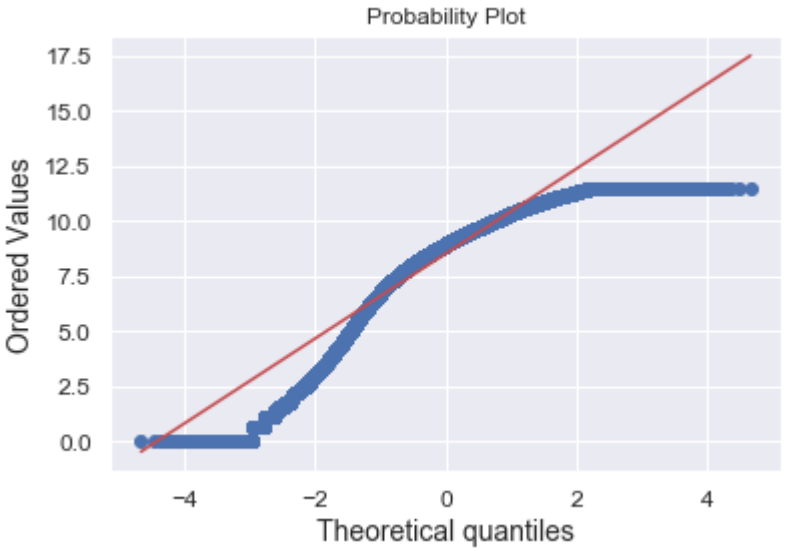
In [48]:
```
train['Weekly_Sales']=np.log(train['Weekly_Sales']+1)
```

In [49]:
```
sns.distplot(train.Weekly_Sales)
```

Out[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc3a8588>
```

In [ ]:
```
## Use Box cox transformation need to plot.
```

In [50]:
```
# Now check residuals
from scipy import stats
import pylab

stats.probplot(train.Weekly_Sales, dist="norm", plot=pylab )
pylab.show()
```



Classification & Accuracy

Define training and testing set

In [51]:
```
#### train X= Exery thing except Weekly_Sales
train_X=train.drop(['Weekly_Sales','Date'], axis=1)

#### train Y= Only Weekly_Sales
train_y=train['Weekly_Sales']

#### Test_X
test_X=test.drop('Date',axis=1).copy()

train_X.shape, train_y.shape, test_X.shape
```

Out[51]: ((420212, 15), (420212,), (115064, 15))

Building models & comparing their RMSE values

1.Linear Regression

In [52]:
```
## Method 1..
clf = LinearRegression()
clf.fit(train_X, train_y)
y_pred_linear=clf.predict(test_X)
acc_linear=round( clf.score(train_X, train_y) * 100, 2)
print ('scorbe:'+str(acc_linear) + ' percent')
```

```
scorbe:11.03 percent
```

In [59]:
```
import statsmodels.api as sm
```

In [60]:
```
train_x = sm.add_constant(train_X)
lm=sm.OLS(train_y,train_X).fit()
```

In [61]:
```
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Weekly_Sales   R-squared:
0.110
Model:                            OLS   Adj. R-squared:
0.110
Method:                 Least Squares   F-statistic:
3720.
Date:                Tue, 03 Sep 2019   Prob (F-statistic):
0.00
Time:                        21:56:39   Log-Likelihood:
-8.6782e+05
No. Observations:              420212   AIC:
1.736e+06
```

```
Df Residuals:                   420197   BIC:
1.736e+06
Df Model:                           14
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
Store             -0.0134      0.000    -56.412      0.000      -0.014
-0.013
Dept               0.0016   9.65e-05     16.264      0.000       0.001
0.002
IsHoliday         -0.0520      0.012     -4.169      0.000      -0.076
-0.028
Type               0.1135      0.008     14.836      0.000       0.099
0.129
Size            1.084e-05   8.38e-08    129.421      0.000    1.07e-05
 1.1e-05
Temperature       -0.0036      0.000    -21.059      0.000      -0.004
-0.003
Fuel_Price         0.0329      0.008      4.130      0.000       0.017
0.049
MarkDown1       1.071e-05   1.02e-06     10.539      0.000    8.72e-06
1.27e-05
MarkDown2      -8.538e-07   6.17e-07     -1.384      0.166   -
2.06e-06    3.56e-07
MarkDown3       3.695e-06   5.59e-07      6.613      0.000     2.6e-06
4.79e-06
MarkDown4      -6.834e-06   1.43e-06     -4.794      0.000   -
9.63e-06   -4.04e-06
Date_dayofweek    35.7844      3.199     11.187      0.000      29.515
42.054
Date_month         0.0160      0.001     16.088      0.000       0.014
0.018
Date_year         -0.0676      0.006    -10.621      0.000      -0.080
-0.055
Date_day          -0.0004      0.000     -1.049      0.294      -0.001
 0.000
==============================================================================
Omnibus:                    79726.360   Durbin-Watson:
1.429
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
156534.468
Skew:                          -1.156   Prob(JB):
0.00
Kurtosis:                       4.896   Cond. No.
1.63e+08
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the er
rors is correctly specified.
[2] The condition number is large, 1.63e+08. This might indicat
e that there are
strong multicollinearity or other numerical problems.
```

2. Random Forest

In [53]:
```
clf = RandomForestRegressor(n_estimators=100)
clf.fit(train_X, train_y)
y_pred_rf=clf.predict(test_X)
acc_rf= round(clf.score(train_X, train_y) * 100, 2)
print ("Accuracy: %i %% \n"%acc_rf)
```

```
Accuracy: 99 %
```

3. Decision Tree

In [54]:
```
clf=DecisionTreeRegressor()
clf.fit(train_X, train_y)
y_pred_dt= clf.predict(test_X)
acc_dt = round( clf.score(train_X, train_y) * 100, 2)
print (str(acc_dt) + ' percent')
```

```
100.0 percent
```

Comparing Models

Let's compare the accuracy score of all the regression models used above.

In [55]:
```
models = pd.DataFrame({
    'Model': ['Linear Regression','Random Forest','Decision Tre
e'],

    'Score': [acc_linear, acc_rf,acc_dt]
    })

models.sort_values(by='Score', ascending=False)
```

Out[55]:

	Model	Score
2	Decision Tree	100.00
1	Random Forest	99.63
0	Linear Regression	11.03

Predicting Sales value for test data based on highest score model.

In [57]:
```
# Prediction value using Random Forest model..
submission = pd.DataFrame({
```

Nov 10, 2019 (http://www.datasciencelovers.com/2019/11/)

Credit Card Segmentation (http://www.datasciencelovers.com/machine-learning-projects/credit-card-segmentation/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/credit-card-segmentation/) Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/)** Tag **clustering (http://www.datasciencelovers.com/tag/clustering/), credit card segmentation (http://www.datasciencelovers.com/tag/credit-card-segmentation/), machine learning (http://www.datasciencelovers.com/tag/machine-learning/), unsupervised learning (http://www.datasciencelovers.com/tag/unsupervised-learning/)**

Data Available:

- CC GENERAL.csv

Business Context:

A Bank wants to develop a customer segmentation to define marketing strategy. The sample dataset summarizes the usage behaviour of about 9000 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioural variables.

Business Requirements:

Advanced data preparation: Build an enriched customer profile by deriving “intelligent” KPIs such as:

- Monthly average purchase and cash advance amount
- Purchases by type (one-off, instalments)
- Average amount per purchase and cash advance transaction,
- Limit usage (balance to credit limit ratio),
- Payments to minimum payments ratio etc.
- Advanced reporting: Use the derived KPIs to gain insight on the customer profiles.
- Identification of the relationships/ affinities between services.
- Clustering: Apply a data reduction technique factor analysis for variable reduction technique and a clustering algorithm to reveal the behavioural segments of credit card holders
- Identify cluster characteristics of the cluster using detailed profiling.
- Provide the strategic insights and implementation of strategies for given set of cluster characteristics.

Data Dictionary:

- **CUST_ID**: Credit card holder ID
- **BALANCE**: Monthly average balance (based on daily balance averages)
- **BALANCE_FREQUENCY:** Ratio of last 12 months with balance
- **PURCHASES:** Total purchase amount spent during last 12 months
- **ONEOFF_PURCHASES:** Total amount of one-off purchases
- **INSTALLMENTS_PURCHASES**: Total amount of installment purchases
- **CASH_ADVANCE**: Total cash-advance amount
- **PURCHASES_ FREQUENCY**: Frequency of purchases (Percent of months with at least one purchase)
- **ONEOFF_PURCHASES_FREQUENCY:** Frequency of one-off-purchases
  PURCHASES_INSTALLMENTS_FREQUENCY: Frequency of installment purchases

- **CASH_ADVANCE_ FREQUENCY**: Cash-Advance frequency
- **AVERAGE_PURCHASE_TRX**: Average amount per purchase transaction
- **CASH_ADVANCE_TRX**: Average amount per cash-advance transaction
- **PURCHASES_TRX**: Average amount per purchase transaction
- **CREDIT_LIMIT**: Credit limit
- **PAYMENTS:** Total payments (due amount paid by the customer to decrease their statement balance) in the period
- **MINIMUM_PAYMENTS:** Total minimum payments due in the period.
- **PRC_FULL_PAYMEN**: Percentage of months with full payment of the due statement balance
- **TENURE**: Number of months as a customer

Let’s develop a machine learning model for further analysis.

Credit Card segmentation with Machine Learning

Import Library

In [1]:
```
#Basic python library which need to import
import pandas as pd
import numpy as np

#Date stuff
from datetime import datetime
from datetime import timedelta

#Library for Nice graphing
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as sn
%matplotlib inline

#Library for statistics operation
import scipy.stats as stats

# Date Time library
from datetime import datetime

#Machine learning Library
import statsmodels.api as sm
from sklearn import metrics
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error, mean_squared_er
ror

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Settings
pd.set_option('display.max_columns', None)
np.set_printoptions(threshold=np.nan)
np.set_printoptions(precision=3)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_valida
tion.py:41: DeprecationWarning: This module was deprecated in v
ersion 0.18 in favor of the model_selection module into which a
ll the refactored classes and functions are moved. Also note th
at the interface of the new CV iterators are different from tha
```

```
at the interface of the new CV iterators are different from tha
t of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\wei
ght_boosting.py:29: DeprecationWarning: numpy.core.umath_tests
is an internal NumPy module and should not be imported. It will
be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

Load data

In [2]:
```
# reading data into dataframe
credit= pd.read_csv("CC_GENERAL.csv")
```

Information about data set

In [3]:
```
credit.head()
```

Out[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURC
0	C10001	40.900749	0.818182	95.40	
1	C10002	3202.467416	0.909091	0.00	
2	C10003	2495.148862	1.000000	773.17	
3	C10004	1666.670542	0.636364	1499.00	
4	C10005	817.714335	1.000000	16.00	

In [4]:
```
credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
CUST_ID                             8950 non-null object
BALANCE                             8950 non-null float64
BALANCE_FREQUENCY                   8950 non-null float64
PURCHASES                           8950 non-null float64
ONEOFF_PURCHASES                    8950 non-null float64
INSTALLMENTS_PURCHASES              8950 non-null float64
CASH_ADVANCE                        8950 non-null float64
PURCHASES_FREQUENCY                 8950 non-null float64
ONEOFF_PURCHASES_FREQUENCY          8950 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY    8950 non-null float64
CASH_ADVANCE_FREQUENCY              8950 non-null float64
CASH_ADVANCE_TRX                    8950 non-null int64
PURCHASES_TRX                       8950 non-null int64
CREDIT_LIMIT                        8949 non-null float64
PAYMENTS                            8950 non-null float64
MINIMUM_PAYMENTS                    8637 non-null float64
PRC_FULL_PAYMENT                    8950 non-null float64
TENURE                              8950 non-null int64
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [5]:
```
# Find the total number of missing values in the dataframe
print ("\nMissing values :  ", credit.isnull().sum().values.sum
())

# printing total numbers of Unique value in the dataframe.
print ("\nUnique values :  \n",credit.nunique())
```

```
Missing values :   314

Unique values :
 CUST_ID                             8950
BALANCE                             8871
BALANCE_FREQUENCY                     43
PURCHASES                           6203
ONEOFF_PURCHASES                    4014
INSTALLMENTS_PURCHASES              4452
CASH_ADVANCE                        4323
PURCHASES_FREQUENCY                   47
ONEOFF_PURCHASES_FREQUENCY            47
PURCHASES_INSTALLMENTS_FREQUENCY      47
CASH_ADVANCE_FREQUENCY                54
CASH_ADVANCE_TRX                      65
PURCHASES_TRX                        173
CREDIT_LIMIT                         205
PAYMENTS                            8711
MINIMUM_PAYMENTS                    8636
PRC_FULL_PAYMENT                      47
TENURE                                 7
dtype: int64
```

In [6]:
```
credit.shape
```

Out[6]: (8950, 18)

In [7]:
```
# Intital descriptive analysis of data.
credit.describe()
```

Out[7]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHAS
count	8950.000000	8950.000000	8950.000000	8950.000(
mean	1564.474828	0.877271	1003.204834	592.4373
std	2081.531879	0.236904	2136.634782	1659.8879
min	0.000000	0.000000	0.000000	0.000(
25%	128.281915	0.888889	39.635000	0.000(
50%	873.385231	1.000000	361.280000	38.000(
75%	2054.140036	1.000000	1110.130000	577.405(
max	19043.138560	1.000000	49039.570000	40761.250(

a) Missing Value Treatment

- Since there are missing values in the data so we are imputing th
em with median

em with median.

In [8]:
```
credit.isnull().any()
```

Out[8]:
```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                         True
PAYMENTS                            False
MINIMUM_PAYMENTS                     True
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

In [9]:
```
# CREDIT_LIMIT  and MINIMUM_PAYMENTS has missing values so we ne
ed to remove with median.

credit['CREDIT_LIMIT'].fillna(credit['CREDIT_LIMIT'].median(),in
place=True)

credit['CREDIT_LIMIT'].count()


credit['MINIMUM_PAYMENTS'].median()
credit['MINIMUM_PAYMENTS'].fillna(credit['MINIMUM_PAYMENTS'].med
ian(),inplace=True)
```

In [13]:
```
# Now again check the missing values.

credit.isnull().any()
```

Out[13]:
```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                        False
PAYMENTS                            False
MINIMUM_PAYMENTS                    False
PRC_FULL_PAYMENT                    False
```

```
PRC_FULL_PAYMENT                  False
TENURE                            False
dtype: bool
```

Deriving New KPI

1. Monthly average purchase and cash advance amount

Monthly_avg_purchase

In [10]:
```
credit['Monthly_avg_purchase']=credit['PURCHASES']/credit['TENURE']
```

In [12]:
```
print(credit['Monthly_avg_purchase'].head(),'\n ',
credit['TENURE'].head(),'\n', credit['PURCHASES'].head())
```

```
0      7.950000
1      0.000000
2     64.430833
3    124.916667
4      1.333333
Name: Monthly_avg_purchase, dtype: float64
  0    12
1    12
2    12
3    12
4    12
Name: TENURE, dtype: int64
 0      95.40
1       0.00
2     773.17
3    1499.00
4      16.00
Name: PURCHASES, dtype: float64
```

Monthly_cash_advance Amount

In [13]:
```
credit['Monthly_cash_advance']=credit['CASH_ADVANCE']/credit['TENURE']
```

In [14]:
```
credit[credit['ONEOFF_PURCHASES']==0]['ONEOFF_PURCHASES'].count()
```

Out[14]:
```
4302
```

2- Purchases by type (one-off, installments)

- To find what type of purchases customers are making on credit card

In [45]:
```
credit.loc[:,['ONEOFF_PURCHASES','INSTALLMENTS_PURCHASES']]
```

Out[45]:

	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES
0	0.00	95.40
1	0.00	0.00
2	773.17	0.00
3	1499.00	0.00
4	16.00	0.00
5	0.00	1333.28
6	6402.63	688.38
7	0.00	436.20
8	661.49	200.00
9	1281.60	0.00
10	0.00	920.12
11	1492.18	0.00
12	2500.23	717.76
13	419.96	1717.97
14	0.00	0.00
15	0.00	1611.70
16	0.00	0.00
17	0.00	519.00
18	166.00	338.35
19	0.00	398.64
20	0.00	176.68
21	5910.04	449.91
22	0.00	815.90
23	3454.56	793.79
24	0.00	0.00
25	0.00	399.60
26	102.00	0.00
27	0.00	233.28
28	204.55	182.50
29	0.00	100.00
...	...	...
8920	0.00	0.00
8921	0.00	57.42
8922	0.00	145.98
8923	939.09	959.79
8924	74.00	0.00
8925	0.00	418.59
8926	0.00	580.00
8927	147.80	167.40

8928	0.00	500.00
8929	0.00	0.00
8930	0.00	84.00
8931	0.00	235.80
8932	0.00	180.00
8933	255.62	363.98
8934	0.00	110.50
8935	0.00	465.90
8936	0.00	712.50
8937	0.00	0.00
8938	0.00	0.00
8939	734.40	0.00
8940	0.00	591.24
8941	0.00	214.55
8942	0.00	113.28
8943	20.90	0.00
8944	1012.73	0.00
8945	0.00	291.12
8946	0.00	300.00
8947	0.00	144.40
8948	0.00	0.00
8949	1093.25	0.00

8950 rows × 2 columns

Find customers ONEOFF_PURCHASES and INSTALLMENTS_PURCHASES details

In [16]:
```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
```

Out[16]: (2042, 20)

In [17]:
```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
```

Out[17]: (2774, 20)

In [19]:
```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
```

Out[19]: (1874, 20)

In [20]:
```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
```

Out[20]: (2260, 20)

As per above detail we found out that there are 4 types of purchase behaviour in the data set. So we need to derive a categorical variable based on their behaviour

In [21]:
```
def purchase(credit):
    if (credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0):
        return 'none'
    if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0):
         return 'both_oneoff_installment'
    if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0):
        return 'one_off'
    if (credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0):
        return 'istallment'
```

In [22]:
```
credit['purchase_type']=credit.apply(purchase,axis=1)
```

In [23]:
```
credit['purchase_type'].value_counts()
```

Out[23]:
```
both_oneoff_installment    2774
istallment                 2260
none                       2042
one_off                    1874
Name: purchase_type, dtype: int64
```

4. Limit_usage (balance to credit limit ratio) credit card utilization

- Lower value implies cutomers are maintaing thier balance properly. Lower value means good credit score

In [52]:
```
credit['limit_usage']=credit.apply(lambda x: x['BALANCE']/x['CREDIT_LIMIT'], axis=1)
```

In [53]:
```
credit['limit_usage'].head()
```

Out[53]:
```
0    0.040901
1    0.457495
2    0.332687
3    0.222223
4    0.681429
Name: limit_usage, dtype: float64
```

5- Payments to minimum payments ratio etc.

In [54]:
```
credit['PAYMENTS'].isnull().any()
credit['MINIMUM_PAYMENTS'].isnull().value_counts()
```

Out[54]:
```
False    8950
Name: MINIMUM_PAYMENTS, dtype: int64
```

In [55]:
```
credit['MINIMUM_PAYMENTS'].describe()
```

In [55]:
```
credit['MINIMUM_PAYMENTS'].describe()
```

Out[55]:
```
count     8950.000000
mean       844.906767
std       2332.792322
min          0.019163
25%        170.857654
50%        312.343947
75%        788.713501
max      76406.207520
Name: MINIMUM_PAYMENTS, dtype: float64
```

In [56]:
```
credit['payment_minpay']=credit.apply(lambda x:x['PAYMENTS']/x
['MINIMUM_PAYMENTS'],axis=1)
```

In [57]:
```
credit['payment_minpay']
```

Out[57]:
```
0        1.446508
1        3.826241
2        0.991682
3        0.000000
4        2.771075
5        0.581601
6       32.081820
7        1.276357
8        2.206280
9       11.612605
10       0.498597
11       4.536309
12       1.240830
13       6.593552
14       0.813816
15       0.944800
16       1.040881
17       3.477861
18       2.311048
19       0.084551
20       0.016454
21       1.251952
22      27.332818
23       6.649970
24       0.858615
25       1.320852
26       0.916592
27       1.261155
28       0.971251
29       1.006784
           ...
8920     0.234940
8921     3.304997
8922     0.659354
8923     1.985987
8924     4.856447
8925     5.191574
8926    12.226247
8927     2.678503
8928     6.184871
8929     0.000000
8930     1.385733
8931     2.112821
8932     1.596541
8933     0.314676
```

```
8934     2.448965
8935     0.000000
8936     6.972789
8937     1.936734
8938    64.055621
8939     0.653714
8940     5.745025
8941     1.120950
8942     1.095102
8943     1.348973
8944     0.000000
8945     6.660231
8946     0.883197
8947     0.986076
8948     0.942505
8949     0.715439
Name: payment_minpay, Length: 8950, dtype: float64
```

Extreme value Treatment

- Since there are variables having extreme values so I am doing log-transformation on the dataset to remove outlier effect

In [58]:
```
# log tranformation
cr_log=credit.drop(['CUST_ID','purchase_type'],axis=1).applymap
(lambda x: np.log(x+1))
```

In [59]:
```
cr_log.describe()
```

Out[59]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASE
count	8950.000000	8950.000000	8950.000000	8950.00000
mean	6.161637	0.619940	4.899647	3.20427
std	2.013303	0.148590	2.916872	3.24636
min	0.000000	0.000000	0.000000	0.00000
25%	4.861995	0.635989	3.704627	0.00000
50%	6.773521	0.693147	5.892417	3.66356
75%	7.628099	0.693147	7.013133	6.36027
max	9.854515	0.693147	10.800403	10.61551

In [60]:
```
col=['BALANCE','PURCHASES','CASH_ADVANCE','TENURE','PAYMENTS','M
INIMUM_PAYMENTS','PRC_FULL_PAYMENT','CREDIT_LIMIT']
cr_pre=cr_log[[x for x in cr_log.columns if x not in col ]]
```

In [61]:
```
cr_pre.columns
```

Out[61]:
```
Index(['BALANCE_FREQUENCY', 'ONEOFF_PURCHASES', 'INSTALLMENTS_P
URCHASES',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUE
NCY',
```

```
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'Monthly_avg_purcha
se',
       'Monthly_cash_advance', 'limit_usage', 'payment_minpa
y'],
      dtype='object')
```

In [62]:
```
cr_log.columns
```

Out[62]:
```
Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PUR
CHASES',
       'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FRE
QUENCY',
       'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FR
EQUENCY',
       'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES
_TRX',
       'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FUL
L_PAYMENT',
       'TENURE', 'Monthly_avg_purchase', 'Monthly_cash_advanc
e', 'limit_usage',
       'payment_minpay'],
      dtype='object')
```

Insights from KPIs

Average payment_minpayment ratio for each purchse type.

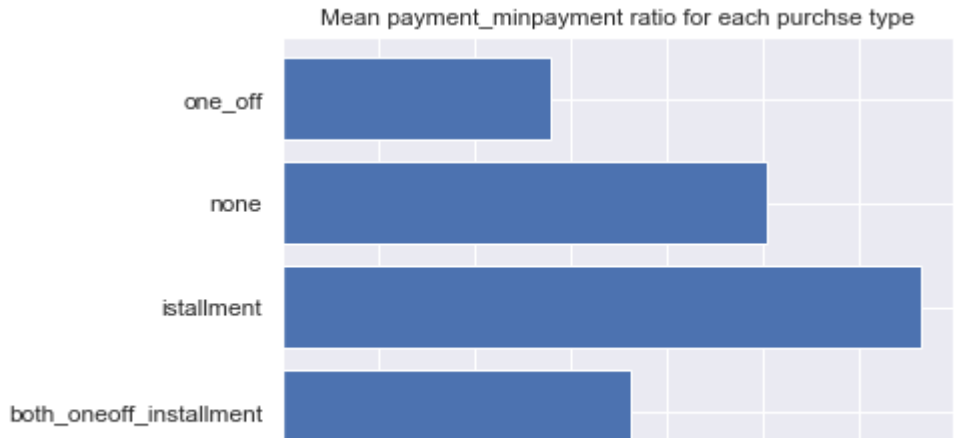
In [63]:
```
x=credit.groupby('purchase_type').apply(lambda x: np.mean(x['pay
ment_minpay']))
type(x)
x.values
```

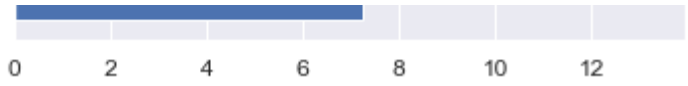
Out[63]:
```
array([ 7.237, 13.259, 10.087,  5.571])
```

In [71]:
```
ax.barh?
```

In [72]:
```
fig,ax=plt.subplots()
ax.barh(y=range(len(x)), width=x.values,align='center')
ax.set(yticks= np.arange(len(x)),yticklabels = x.index);
plt.title('Mean payment_minpayment ratio for each purchse type')
```

Out[72]:
```
Text(0.5, 1.0, 'Mean payment_minpayment ratio for each purchse
type')
```





In [73]:
```
credit.describe()
```

Out[73]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHAS
count	8950.000000	8950.000000	8950.000000	8950.000(
mean	1564.474828	0.877271	1003.204834	592.4373
std	2081.531879	0.236904	2136.634782	1659.8879
min	0.000000	0.000000	0.000000	0.000(
25%	128.281915	0.888889	39.635000	0.000(
50%	873.385231	1.000000	361.280000	38.000(
75%	2054.140036	1.000000	1110.130000	577.405(
max	19043.138560	1.000000	49039.570000	40761.250(

customers with installment purchases are paying dues

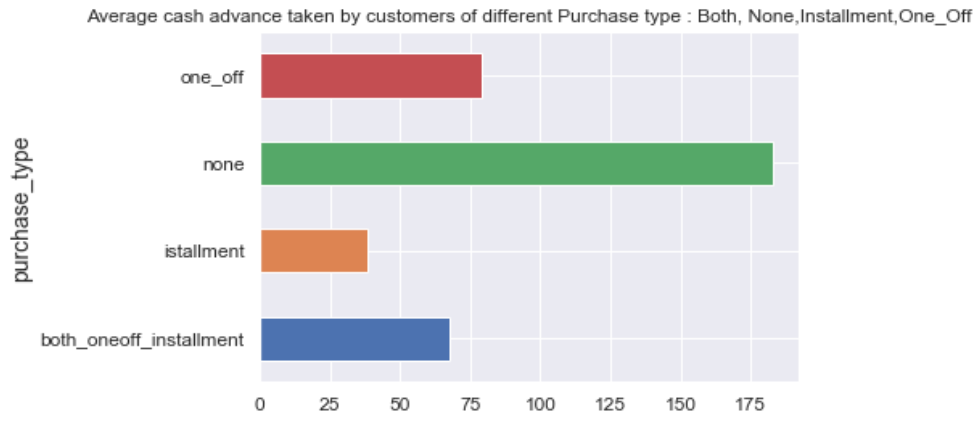
In [77]:
```
credit[credit['purchase_type']=='n']
```

Out[77]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHA

In [78]:
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['Monthly_cash_advance'])).plot.barh()

plt.title('Average cash advance taken by customers of different Purchase type : Both, None,Installment,One_Off')
```

Out[78]:
```
Text(0.5, 1.0, 'Average cash advance taken by customers of different Purchase type : Both, None,Installment,One_Off')
```

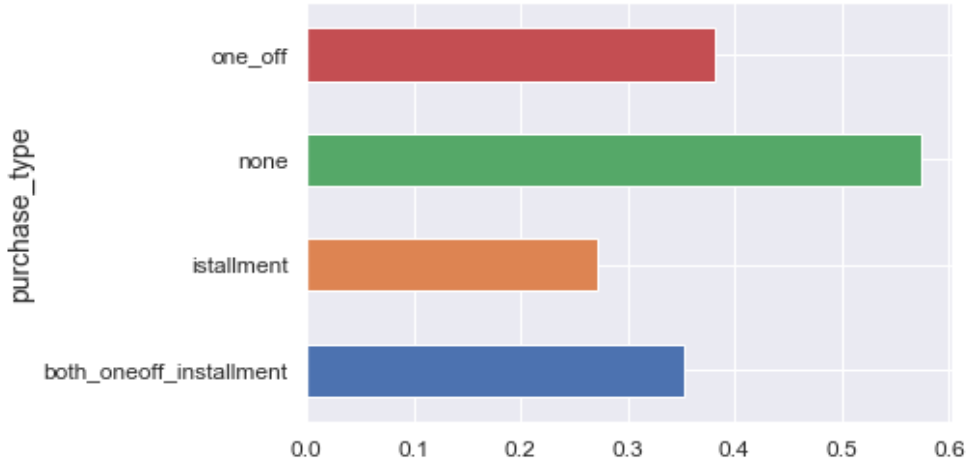


Customers who don't do either one-off or installment purchases take more cash on advance

In [79]:
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit
```

In [79]:
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage'])).plot.barh()
```

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0xc881a20>



Original dataset with categorical column converted to number type.

In [139]:
```
cre_original=pd.concat([credit,pd.get_dummies(credit['purchase_type'])],axis=1)
```

Preparing Machine learning algorithm

We do have some categorical data which need to convert with the help of dummy creation

In [80]:
```
# creating Dummies for categorical variable
cr_pre['purchase_type']=credit.loc[:,'purchase_type']
pd.get_dummies(cr_pre['purchase_type'])
```

Out[80]:

	both_oneoff_installment	istallment	none	one_off
0	0	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	0	0	1
4	0	0	0	1
5	0	1	0	0
6	1	0	0	0
7	0	1	0	0
8	1	0	0	0
9	0	0	0	1
10	0	1	0	0
11	0	0	0	1
12	1	0	0	0

13	1	0	0	0
14	0	0	1	0
15	0	1	0	0
16	0	0	1	0
17	0	1	0	0
18	1	0	0	0
19	0	1	0	0
20	0	1	0	0
21	1	0	0	0
22	0	1	0	0
23	1	0	0	0
24	0	0	1	0
25	0	1	0	0
26	0	0	0	1
27	0	1	0	0
28	1	0	0	0
29	0	1	0	0
...	...	...	...	...
8920	0	0	1	0
8921	0	1	0	0
8922	0	1	0	0
8923	1	0	0	0
8924	0	0	0	1
8925	0	1	0	0
8926	0	1	0	0
8927	1	0	0	0
8928	0	1	0	0
8929	0	0	1	0
8930	0	1	0	0
8931	0	1	0	0
8932	0	1	0	0
8933	1	0	0	0
8934	0	1	0	0
8935	0	1	0	0
8936	0	1	0	0
8937	0	0	1	0
8938	0	0	1	0
8939	0	0	0	1
8940	0	1	0	0
8941	0	1	0	0

8942	0	1	0	0
8943	0	0	0	1
8944	0	0	0	1
8945	0	1	0	0
8946	0	1	0	0
8947	0	1	0	0
8948	0	0	1	0
8949	0	0	0	1

8950 rows × 4 columns

Now merge the created dummy with the original data frame

In [81]:
```
cr_dummy=pd.concat([cr_pre,pd.get_dummies(cr_pre['purchase_type'])],axis=1)
```

In [82]:
```
l=['purchase_type']
```

In [83]:
```
cr_dummy=cr_dummy.drop(l,axis=1)
cr_dummy.isnull().any()
```

Out[83]:
```
BALANCE_FREQUENCY                   False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
Monthly_avg_purchase                False
Monthly_cash_advance                False
limit_usage                         False
payment_minpay                      False
both_oneoff_installment             False
istallment                          False
none                                False
one_off                             False
dtype: bool
```

In [84]:
```
cr_dummy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 17 columns):
BALANCE_FREQUENCY                   8950 non-null float64
ONEOFF_PURCHASES                    8950 non-null float64
INSTALLMENTS_PURCHASES              8950 non-null float64
PURCHASES_FREQUENCY                 8950 non-null float64
ONEOFF_PURCHASES_FREQUENCY          8950 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY    8950 non-null float64
CASH_ADVANCE_FREQUENCY              8950 non-null float64
CASH_ADVANCE_TRX                    8950 non-null float64
PURCHASES_TRX                       8950 non-null float64
```

```
Monthly_avg_purchase                8950 non-null float64
Monthly_cash_advance                8950 non-null float64
limit_usage                         8950 non-null float64
payment_minpay                      8950 non-null float64
both_oneoff_installment             8950 non-null uint8
istallment                          8950 non-null uint8
none                                8950 non-null uint8
one_off                             8950 non-null uint8
dtypes: float64(13), uint8(4)
memory usage: 944.0 KB
```

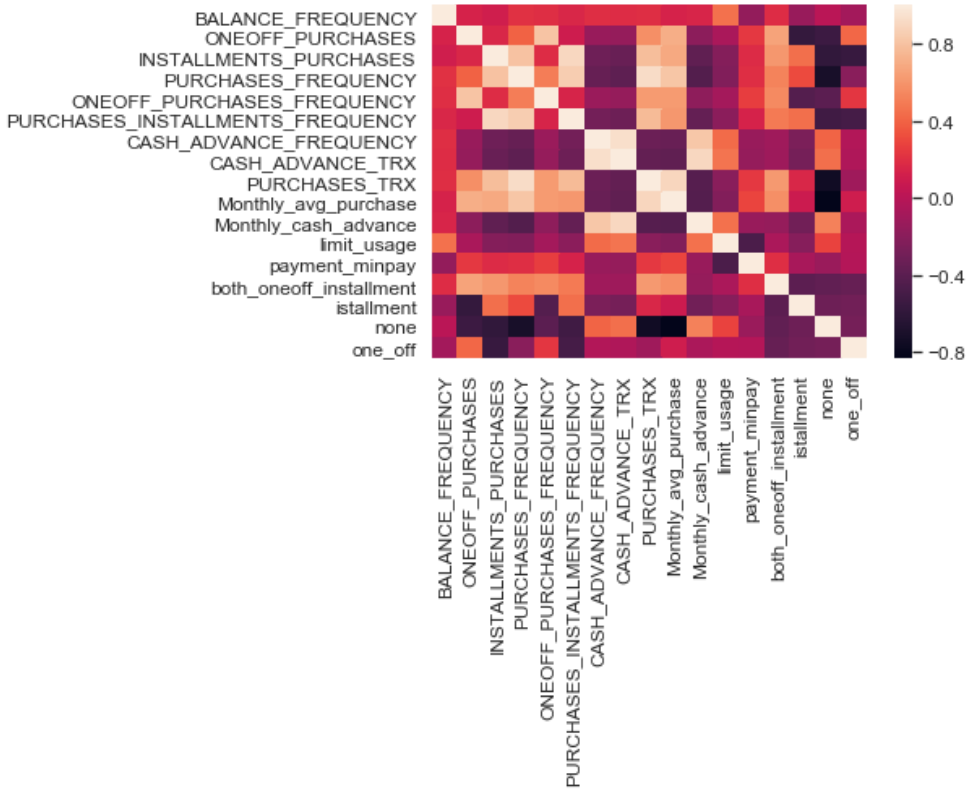
In [85]:
```
cr_dummy.head(3)
```

Out[85]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES
0	0.597837	0.000000	4.568506
1	0.646627	0.000000	0.000000
2	0.693147	6.651791	0.000000

In [86]:
```
sns.heatmap(cr_dummy.corr())
```

Out[86]: `<matplotlib.axes._subplots.AxesSubplot at 0xcaf1cc0>`



- Heat map shows that many features are co-related so applying dimensionality reduction will help negating multi-colinearity in data

- Before applying PCA we will standardize data to avoid effect of scale on our result. Centering and Scaling will make all features with equal weight.

Standardizing data

Standardrizing data

- To put data on the same scale

```
In [88]: from sklearn.preprocessing import  StandardScaler
```

```
In [89]: sc=StandardScaler()
```

```
In [90]: cr_dummy.shape
```

```
Out[90]: (8950, 17)
```

```
In [91]: cr_scaled=sc.fit_transform(cr_dummy)
```

```
In [ ]: cr_scaled
```

Applying PCA

With the help of principal component analysis we will reduce features

```
In [94]: from sklearn.decomposition import PCA
```

```
In [95]: cr_dummy.shape
```

```
Out[95]: (8950, 17)
```

```
In [96]: #We have 17 features so our n_component will be 17.
         pc=PCA(n_components=17)
         cr_pca=pc.fit(cr_scaled)
```

```
In [97]: #Lets check if we will take 17 component then how much varience
         it explain. Ideally it should be 1 i.e 100%
         sum(cr_pca.explained_variance_ratio_)
```

```
Out[97]: 1.0
```

```
In [98]: var_ratio={}
         for n in range(2,18):
             pc=PCA(n_components=n)
             cr_pca=pc.fit(cr_scaled)
             var_ratio[n]=sum(cr_pca.explained_variance_ratio_)
```

```
In [99]: var_ratio
```

```
Out[99]: {2: 0.5826439793960285,
          3: 0.7299379309512699,
          4: 0.8115442762351249,
          5: 0.8770555795291439,
          6: 0.9186492443512616,
          7: 0.9410925256030134,
          8: 0.961611405368306,
          9: 0.9739787081990642,
          10: 0.9835896584630712,
          11: 0.9897248107341952,
          12: 0.9927550009135226,
```

```
 13: 0.9953907562385428,
 14: 0.9979616898169594,
 15: 0.9996360473172955,
 16: 1.0,
 17: 1.0}
```

Since 6 components are explaining about 90% variance so we select 5 components

In [100]:
```
pc=PCA(n_components=6)
```

In [101]:
```
p=pc.fit(cr_scaled)
```

In [102]:
```
cr_scaled.shape
```

Out[102]:
```
(8950, 17)
```

In [103]:
```
p.explained_variance_
```

Out[103]:
```
array([6.836, 3.07 , 2.504, 1.387, 1.114, 0.707])
```

In [104]:
```
np.sum(p.explained_variance_)
```

Out[104]:
```
15.61878226930881
```

In [105]:
```
np.sum(p.explained_variance_)
```

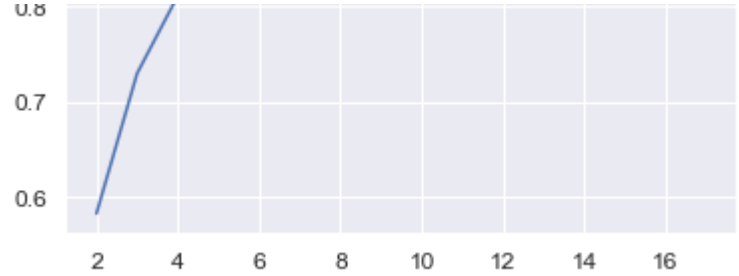
Out[105]:
```
15.61878226930881
```

In [106]:
```
var_ratio
```

Out[106]:
```
{2: 0.5826439793960285,
 3: 0.7299379309512699,
 4: 0.8115442762351249,
 5: 0.8770555795291439,
 6: 0.9186492443512616,
 7: 0.9410925256030134,
 8: 0.961611405368306,
 9: 0.9739787081990642,
 10: 0.9835896584630712,
 11: 0.9897248107341952,
 12: 0.9927550009135226,
 13: 0.9953907562385428,
 14: 0.9979616898169594,
 15: 0.9996360473172955,
 16: 1.0,
 17: 1.0}
```

In [107]:
```
pd.Series(var_ratio).plot()
```

Out[107]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc9842e8>
```



Since 5 components are explaining about 87% variance so we select 5 components

In [108]:
```
cr_scaled.shape
```

Out[108]:
```
(8950, 17)
```

In [109]:
```
pc_final=PCA(n_components=6).fit(cr_scaled)

reduced_cr=pc_final.fit_transform(cr_scaled)
```

In [110]:
```
dd=pd.DataFrame(reduced_cr)
```

In [111]:
```
dd.head()
```

Out[111]:

	0	1	2	3	4	5
0	-0.242841	-2.759668	0.343061	-0.417359	-0.007100	0.019755
1	-3.975652	0.144625	-0.542989	1.023832	-0.428929	-0.572463
2	1.287396	1.508938	2.709966	-1.892252	0.010809	-0.599932
3	-1.047613	0.673103	2.501794	-1.306784	0.761348	1.408986
4	-1.451586	-0.176336	2.286074	-1.624896	-0.561969	-0.675214

So initially we had 17 variables now its 5 so our variable go reduced

In [112]:
```
dd.shape
```

Out[112]:
```
(8950, 6)
```

In [113]:
```
col_list=cr_dummy.columns
```

In [114]:
```
col_list
```

Out[114]:
```
Index(['BALANCE_FREQUENCY', 'ONEOFF_PURCHASES', 'INSTALLMENTS_P
URCHASES',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUE
NCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'Monthly_avg_purcha
se',
       'Monthly_cash_advance', 'limit_usage', 'payment_minpay',
       'both_oneoff_installment', 'istallment', 'none', 'one_of
f'],
      dtype='object')
```

In [115]:
```
pd.DataFrame(pc_final.components_.T, columns=['PC_' +str(i) for i in range(6)],index=col_list)
```

Out[115]:

	PC_0	PC_1	PC_2	
BALANCE_FREQUENCY	0.029707	0.240072	-0.263140	-0.35
ONEOFF_PURCHASES	0.214107	0.406078	0.239165	0.00
INSTALLMENTS_PURCHASES	0.312051	-0.098404	-0.315625	0.08
PURCHASES_FREQUENCY	0.345823	0.015813	-0.162843	-0.07
ONEOFF_PURCHASES_FREQUENCY	0.214702	0.362208	0.163222	0.03
PURCHASES_INSTALLMENTS_FREQUENCY	0.295451	-0.112002	-0.330029	0.02
CASH_ADVANCE_FREQUENCY	-0.214336	0.286074	-0.278586	0.09
CASH_ADVANCE_TRX	-0.229393	0.291556	-0.285089	0.10
PURCHASES_TRX	0.355503	0.106625	-0.102743	-0.05
Monthly_avg_purchase	0.345992	0.141635	0.023986	-0.07
Monthly_cash_advance	-0.243861	0.264318	-0.257427	0.13
limit_usage	-0.146302	0.235710	-0.251278	-0.43
payment_minpay	0.119632	0.021328	0.136357	0.59
both_oneoff_installment	0.241392	0.273676	-0.131935	0.25
istallment	0.082209	-0.443375	-0.208683	-0.19
none	-0.310283	-0.005214	-0.096911	0.24
one_off	-0.042138	0.167737	0.472749	-0.33

So above data gave us eigen vector for each component we had all eigen vector value very small we can remove those variable bur in our case its not.

In [117]:
```
# Factor Analysis : variance explained by each component-
pd.Series(pc_final.explained_variance_ratio_,index=['PC_'+ str(i) for i in range(6)])
```

Out[117]:
```
PC_0    0.402058
PC_1    0.180586
PC_2    0.147294
PC_3    0.081606
PC_4    0.065511
PC_5    0.041594
dtype: float64
```

Clustering

Based on the intuition on type of purchases made by customers and their distinctive behavior exhibited based on the purchase_type (as visualized above in Insights from KPI) , I am starting with 4 clusters.

In [119]:
```
from sklearn.cluster import KMeans
```

In [120]:
```
km_4=KMeans(n_clusters=4,random_state=123)
```

In [121]:
```
km_4.fit(reduced_cr)
```

Out[121]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_ite
r=300,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='au
to',
    random_state=123, tol=0.0001, verbose=0)
```

In [ ]:
```
km_4.labels_
```

In [123]:
```
pd.Series(km_4.labels_).value_counts()
```

Out[123]:
```
3    2769
2    2224
1    2088
0    1869
dtype: int64
```

Here we donot have known k value so we will find the K. To do that we need to take a cluster range between 1 and 21.

Identify cluster Error.

In [124]:
```
cluster_range = range( 1, 21 )
cluster_errors = []

for num_clusters in cluster_range:
    clusters = KMeans( num_clusters )
    clusters.fit( reduced_cr )
    cluster_errors.append( clusters.inertia_ )# clusters.inertia
_ is basically cluster error here.
```

In [125]:
```
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "clu
ster_errors": cluster_errors } )

clusters_df[0:21]
```

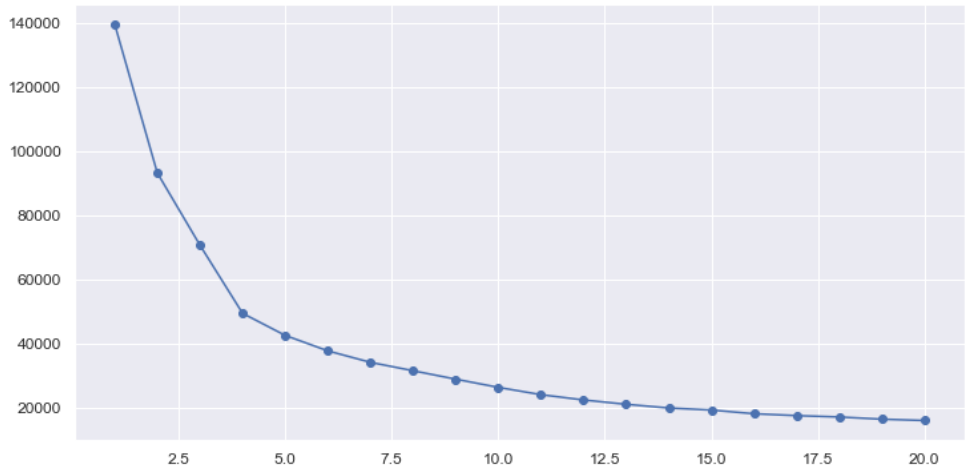
Out[125]:

	num_clusters	cluster_errors
0	1	139772.482528
1	2	93308.123825
2	3	70745.193400
3	4	49446.066485
4	5	42548.525149
5	6	37713.045270
6	7	34124.456465
7	8	31502.720956
8	9	28865.971034

9	10	26302.837007
10	11	24020.031165
11	12	22364.382649
12	13	21006.533766
13	14	19857.234322
14	15	19214.601923
15	16	18057.825042
16	17	17469.334973
17	18	17040.494574
18	19	16356.843904
19	20	15937.614441

In [126]:
```
# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors,
marker = "o" )
```

Out[126]:
```
[<matplotlib.lines.Line2D at 0xc8812e8>]
```



From above graph we will find elbow range. here it is 4,5,6

Silhouette Coefficient

In [127]:
```
from sklearn import metrics
```

In [128]:
```
# calculate SC for K=3 through K=12
k_range = range(2, 21)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=1)
    km.fit(reduced_cr)
    scores.append(metrics.silhouette_score(reduced_cr, km.labels
_))
```

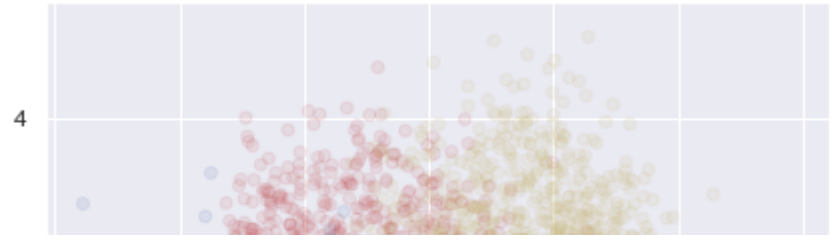
In [129]:
```
scores
```

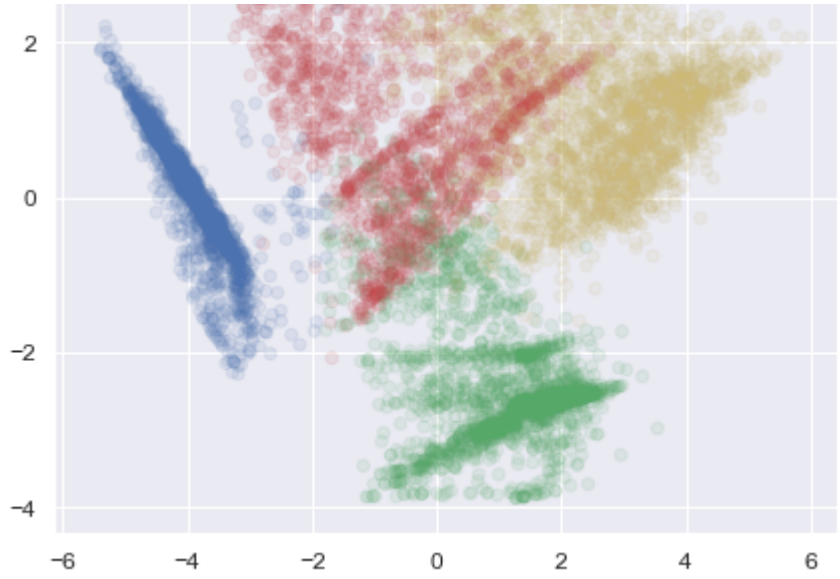
Out[129]:
```
[0.33194521792342696,
 0.3540311511892251,
 0.43708577439659485,
 0.43121145209717765,
 0.4228144914653746,
 0.4022440826179354,
 0.4144537298622617,
 0.3889240713086914,
 0.392999135547462,
 0.3678798374268567,
 0.3669766371659527,
 0.3403473055008083,
 0.3526966868656406,
 0.3439233788957459,
 0.3336956707851422,
 0.32231668200851954,
 0.3322902581427395,
 0.3274102528436649,
 0.3239792346891927]
```

In [130]:
```
# plot the results
plt.plot(k_range, scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.grid(True)
```

In [131]:
```
color_map={0:'r',1:'b',2:'g',3:'y'}
label_color=[color_map[l] for l in km_4.labels_]
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=label_color,cmap
='Spectral',alpha=0.1)
```

Out[131]:
```
<matplotlib.collections.PathCollection at 0xe33b2b0>
```





It is very difficult to draw iddividual plot for cluster, so we will use pair plot which will provide us all graph in one shot. To do that we need to take following steps

In [132]:
```
df_pair_plot=pd.DataFrame(reduced_cr,columns=['PC_' +str(i) for i in range(6)])
```

In [133]:
```
df_pair_plot['Cluster']=km_4.labels_ #Add cluster column in the data frame
```

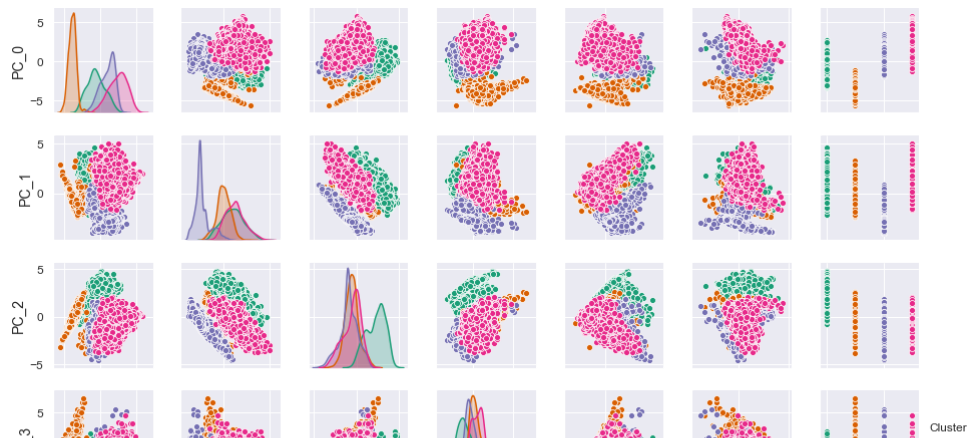
In [134]:
```
df_pair_plot.head()
```

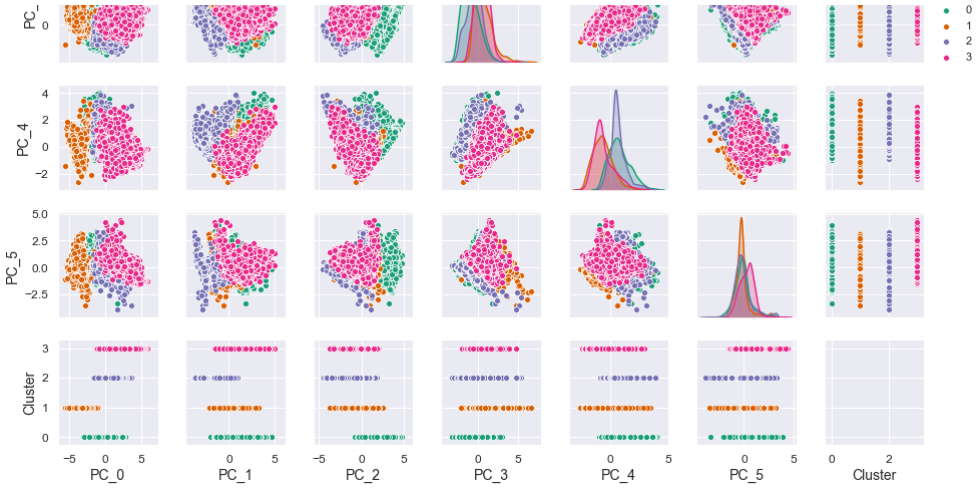
Out[134]:

	PC_0	PC_1	PC_2	PC_3	PC_4	PC_5	Cluster
0	-0.242841	-2.759668	0.343061	-0.417359	-0.007100	0.019755	2
1	-3.975652	0.144625	-0.542989	1.023832	-0.428929	-0.572463	1
2	1.287396	1.508938	2.709966	-1.892252	0.010809	-0.599932	0
3	-1.047613	0.673103	2.501794	-1.306784	0.761348	1.408986	0
4	-1.451586	-0.176336	2.286074	-1.624896	-0.561969	-0.675214	0

In [135]:
```
#pairwise relationship of components on the data
sns.pairplot(df_pair_plot,hue='Cluster', palette= 'Dark2', diag_kind='kde',size=1.85)
```

Out[135]:
```
<seaborn.axisgrid.PairGrid at 0x13534b00>
```





It shows that first two components are able to indentify clusters

Now we have done here with priciple component now we need to come bring our original data frame and we will merge the cluster with them.

To interprate result we need to use our data frame

In [136]:
```
# Key performace variable selection . here i am taking varibales which we will use in derving new KPI.
#We can take all 17 variables but it will be difficult to interprate.So are are selecting less no of variables.

col_kpi=['PURCHASES_TRX','Monthly_avg_purchase','Monthly_cash_advance','limit_usage','CASH_ADVANCE_TRX',
         'payment_minpay','both_oneoff_installment','istallment','one_off','none','CREDIT_LIMIT']
```

In [137]:
```
cr_pre.describe()
```

Out[137]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
count	8950.000000	8950.000000	8950.00(
mean	0.619940	3.204274	3.35:
std	0.148590	3.246365	3.08:
min	0.000000	0.000000	0.00(
25%	0.635989	0.000000	0.00(
50%	0.693147	3.663562	4.49(
75%	0.693147	6.360274	6.15'
max	0.693147	10.615512	10.02'

In [140]:
```
# Conactenating labels found through Kmeans with data
cluster_df_4=pd.concat([cre_original[col_kpi],pd.Series(km_4.labels_,name='Cluster_4')],axis=1)
```

In [141]:
```
cluster_df_4.head()
```

Out[141]:

	PURCHASES_TRX	Monthly_avg_purchase	Monthly_cash_advance	limit_usage
0	2	7.950000	0.000000	0.04090
1	0	0.000000	536.912124	0.45749
2	12	64.430833	0.000000	0.33268
3	1	124.916667	17.149001	0.22222
4	1	1.333333	0.000000	0.68142

In [142]:
```
# Mean value gives a good indication of the distribution of dat
a. So we are finding mean value for each variable for each clust
er
cluster_4=cluster_df_4.groupby('Cluster_4')\
.apply(lambda x: x[col_kpi].mean()).T
cluster_4
```

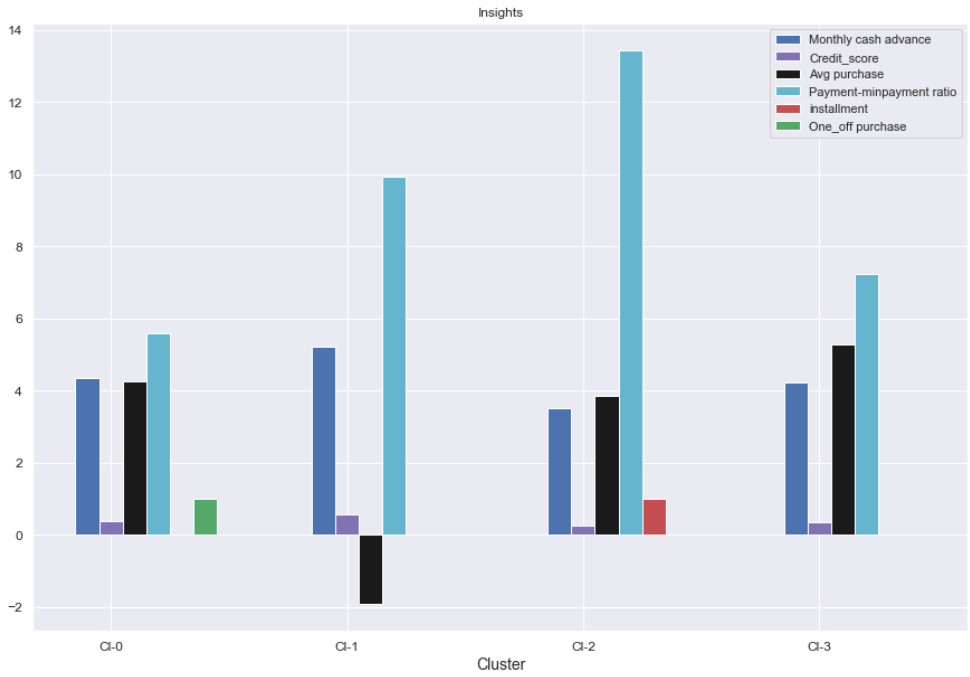
Out[142]:

Cluster_4	0	1	2	3
PURCHASES_TRX	7.127341	0.043582	12.062050	33.013723
Monthly_avg_purchase	69.875917	0.148297	47.626256	193.008043
Monthly_cash_advance	78.098613	186.281319	33.550080	67.466910
limit_usage	0.379761	0.576076	0.264745	0.353591
CASH_ADVANCE_TRX	2.881220	6.540230	1.021133	2.804261
payment_minpay	5.573672	9.936617	13.422420	7.245651
both_oneoff_installment	0.000535	0.001916	0.000000	1.000000
istallment	0.000000	0.017241	1.000000	0.000000
one_off	0.999465	0.002874	0.000000	0.000000
none	0.000000	0.977969	0.000000	0.000000
CREDIT_LIMIT	4519.708481	4055.156450	3338.270406	5736.732730

In [143]:
```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(cluster_4.columns))

cash_advance=np.log(cluster_4.loc['Monthly_cash_advance',:].valu
es)
credit_score=(cluster_4.loc['limit_usage',:].values)
purchase= np.log(cluster_4.loc['Monthly_avg_purchase',:].values)
payment=cluster_4.loc['payment_minpay',:].values
installment=cluster_4.loc['istallment',:].values
one_off=cluster_4.loc['one_off',:].values


bar_width=.10
b1=plt.bar(index,cash_advance,color='b',label='Monthly cash adva
nce',width=bar_width)
b2=plt.bar(index+bar_width,credit_score,color='m',label='Credit_
score',width=bar_width)
b3=plt.bar(index+2*bar_width,purchase,color='k',label='Avg purch
ase',width=bar_width)
b4=plt.bar(index+3*bar_width,payment,color='c',label='Payment-mi
npayment ratio',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='instal
```

```
lment',width=bar_width)
b6=plt.bar(index+5*bar_width,one_off,color='g',label='One_off pu
rchase',width=bar_width)

plt.xlabel("Cluster")
plt.title("Insights")
plt.xticks(index + bar_width, ('Cl-0', 'Cl-1', 'Cl-2', 'Cl-3'))
plt.legend()
```

Out[143]: <matplotlib.legend.Legend at 0x114ffa90>



Insights

Clusters are clearly distinguishing behavior within customers

- Cluster 2 is the group of customers who have highest Monthly_avg purchases and doing both installment as well as one_off purchases, have comparatively good credit score. ***This group is about 31% of the total customer base***
- cluster 1 is taking maximum advance_cash and is paying comparatively less minimum payment and poor credit_score & doing no purchase transaction. ***This group is about 23% of the total customer base***
- Cluster 0 customers are doing maximum One_Off transactions and least payment ratio. ***This group is about 21% of the total customer base***
- Cluster 3 customers have maximum credit score and are paying dues and are doing maximum installment purchases. ***This group is about 25% of the total customer base***

In [149]:
```
# Percentage of each cluster in the total customer
base
s=cluster_df_4.groupby('Cluster_4').apply(lambda x:
```

```
x[ 'Cluster_4' ].value_counts())
print (s),'\n'

per=pd.Series((s.values.astype('float')/ cluster_df
_4.shape[0])*100,name='Percentage')
print ("Cluster -4 "),'\n'
print (pd.concat([pd.Series(s.values,name='Size'),p
er],axis=1))
```

```
Cluster_4
0          0    1869
1          1    2088
2          2    2224
3          3    2769
Name: Cluster_4, dtype: int64
Cluster -4
   Size  Percentage
0  1869   20.882682
1  2088   23.329609
2  2224   24.849162
3  2769   30.938547
```

Finding behaviour with 5 Clusters:

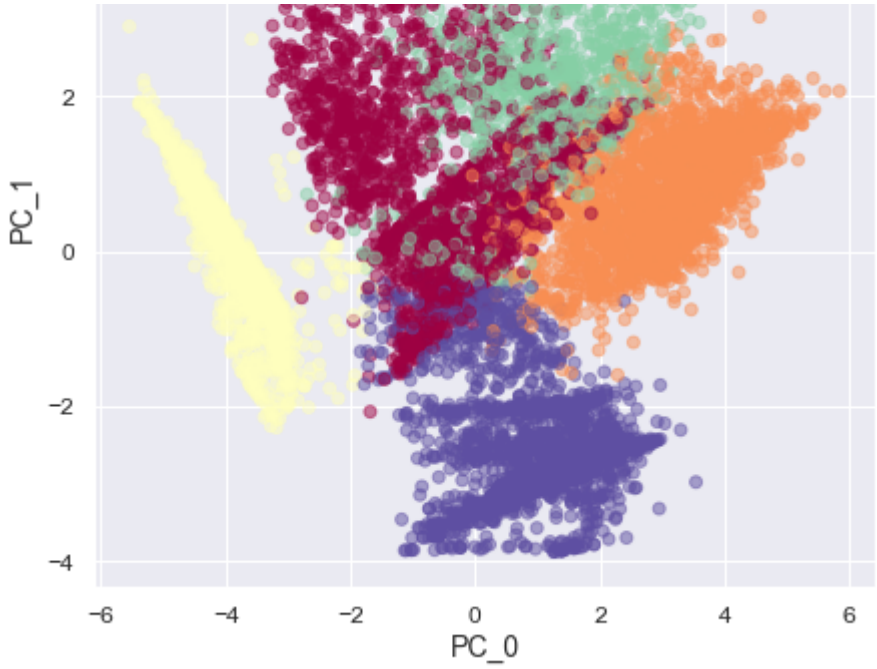
In [ ]:
```
km_5=KMeans(n_clusters=5,random_state=123)
km_5=km_5.fit(reduced_cr)
km_5.labels_
```

In [151]:
```
pd.Series(km_5.labels_).value_counts()
```

Out[151]:
```
4    2149
2    2081
1    1977
0    1862
3     881
dtype: int64
```

In [152]:
```
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=km_5.
labels_,cmap='Spectral',alpha=0.5)
plt.xlabel('PC_0')
plt.ylabel('PC_1')
```

Out[152]:
```
Text(0, 0.5, 'PC_1')
```



In [153]:
```
cluster_df_5=pd.concat([cre_original[col_kpi],pd.Series(km_5.labels_,name='Cluster_5')],axis=1)
```

In [154]:
```
# Finding Mean of features for each cluster
cluster_df_5.groupby('Cluster_5')\
.apply(lambda x: x[col_kpi].mean()).T
```

Out[154]:

Cluster_5	0	1	2	3
PURCHASES_TRX	7.096670	34.587759	0.032196	27.703746
Monthly_avg_purchase	68.917645	210.536468	0.086126	141.584086
Monthly_cash_advance	74.517541	4.040708	185.038534	249.942101
limit_usage	0.377959	0.258931	0.576110	0.600096
CASH_ADVANCE_TRX	2.697637	0.152757	6.448823	10.384790
payment_minpay	5.562287	8.675499	9.963172	3.651686
both_oneoff_installment	0.002148	1.000000	0.000000	0.900114
istallment	0.000000	0.000000	0.015858	0.088536
one_off	0.997852	0.000000	0.002883	0.011351
none	0.000000	0.000000	0.981259	0.000000
CREDIT_LIMIT	4497.951209	5722.970627	4046.692295	5873.041998

Conclusion With 5 clusters :

- we have a group of customers (cluster 2) having highest avergae purchases but there is Cluster 4 also having highest cash advance & secong highest purchase behaviour but their type of purchases are same.
- Cluster 0 and Cluster 4 are behaving similar in terms of Credit_limit and have cash transactions is on higher side

So we don't have quite distinguishable characteristics with 5 clusters,

In [157]:
```
s1=cluster_df_5.groupby('Cluster_5').apply(lambda x: x['Cluster_5'].value_counts())
print (s1)
```

```
Cluster_5
0          0    1862
1          1    1977
2          2    2081
3          3     881
4          4    2149
Name: Cluster_5, dtype: int64
```

In [158]:
```
# percentage of each cluster

print ("Cluster-5"),'\n'
per_5=pd.Series((s1.values.astype('float')/ cluster_df_5.shape[0])*100,name='Percentage')
print (pd.concat([pd.Series(s1.values,name='Size'),per_5],axis=1))
```

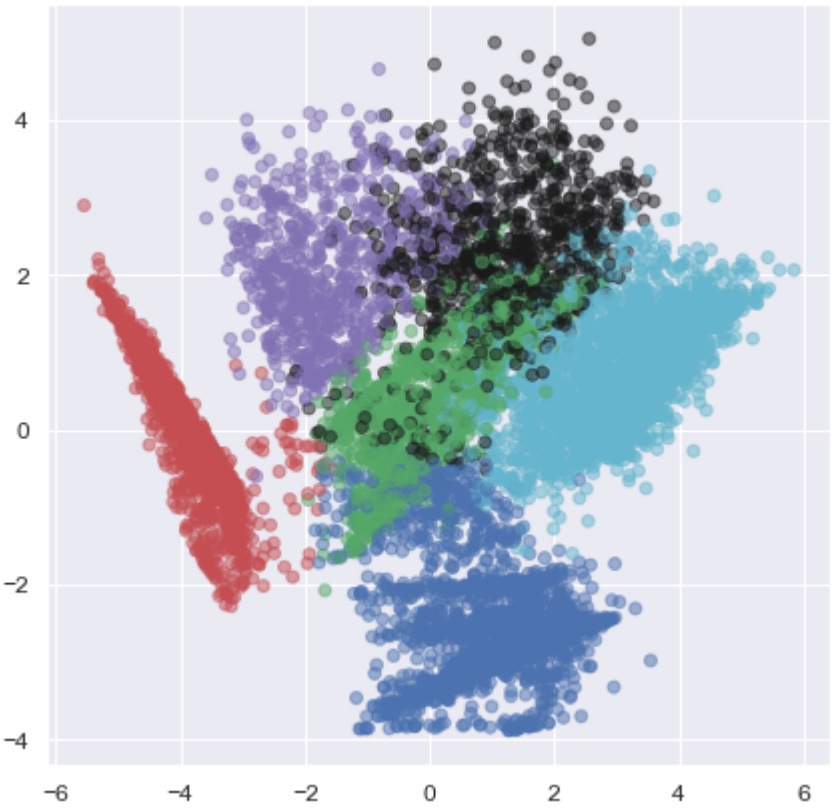
```
Cluster-5
   Size  Percentage
0  1862   20.804469
1  1977   22.089385
2  2081   23.251397
3   881    9.843575
4  2149   24.011173
```

Finding behavior with 6 clusters

In [ ]:
```
km_6=KMeans(n_clusters=6).fit(reduced_cr)
km_6.labels_
```

In [160]:
```
color_map={0:'r',1:'b',2:'g',3:'c',
4:'m',5:'k'}
label_color=[color_map[l] for l in km_6.
labels_]
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr
[:,1],c=label_color,cmap='Spectral',alph
a=0.5)
```

Out[160]:
```
<matplotlib.collections.PathCollection
at 0x11ee4cf8>
```



In [169]:
```
cluster_df_6 = pd.concat([cre_original[c
ol_kpi],pd.Series(km_6.labels_,name='Clu
ster_6')],axis=1)
```

In [170]:
```
six_cluster=cluster_df_6.groupby('Cluste
r_6').apply(lambda x: x[col_kpi].mean
()).T
six_cluster
```

Out[170]:

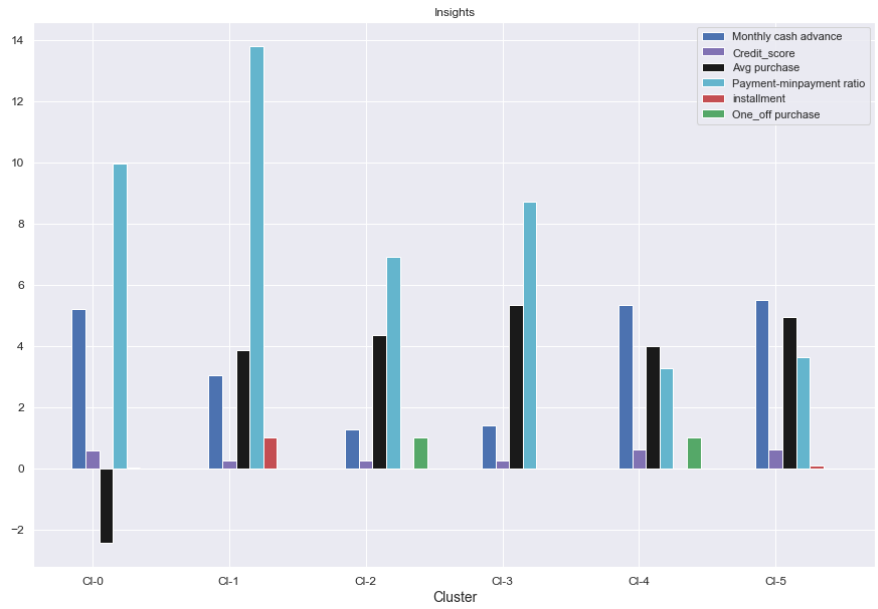
Cluster_6	0	1	2	
PURCHASES_TRX	0.030347	11.905537	7.760575	34.663
Monthly_avg_purchase	0.088891	47.369817	78.585295	211.196
Monthly_cash_advance	184.829434	20.636870	3.603272	4.027

limit_usage	0.575724	0.250011	0.245772	0.258
CASH_ADVANCE_TRX	6.434971	0.550489	0.125212	0.150
payment_minpay	9.976487	13.783426	6.911822	8.702
both_oneoff_installment	0.000000	0.000000	0.006768	1.000
istallment	0.016378	1.000000	0.000000	0.000
one_off	0.000000	0.000000	0.993232	0.000
none	0.983622	0.000000	0.000000	0.000
CREDIT_LIMIT	4047.527296	3228.949923	4471.701020	5735.293

In [171]:
```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(six_cluster.columns))

cash_advance=np.log(six_cluster.loc['Monthly_cash_advance',:].values)
credit_score=(six_cluster.loc['limit_usage',:].values)
purchase= np.log(six_cluster.loc['Monthly_avg_purchase',:].values)
payment=six_cluster.loc['payment_minpay',:].values
installment=six_cluster.loc['istallment',:].values
one_off=six_cluster.loc['one_off',:].values

bar_width=.10
b1=plt.bar(index,cash_advance,color='b',label='Monthly cash advance',width=bar_width)
b2=plt.bar(index+bar_width,credit_score,color='m',label='Credit_score',width=bar_width)
b3=plt.bar(index+2*bar_width,purchase,color='k',label='Avg purchase',width=bar_width)
b4=plt.bar(index+3*bar_width,payment,color='c',label='Payment-minpayment ratio',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=ba
```

```
r_width)
b6=plt.bar(index+5*bar_width,one_off,col
or='g',label='One_off purchase',width=ba
r_width)

plt.xlabel("Cluster")
plt.title("Insights")
plt.xticks(index + bar_width, ('Cl-0',
'Cl-1', 'Cl-2', 'Cl-3','Cl-4','Cl-5'))

plt.legend()
```

Out[171]:
```
<matplotlib.legend.Legend at 0x11f3e5c0
>
```



In [172]:
```
cash_advance=np.log(six_cluster.loc['Mon
thly_cash_advance',:].values)
credit_score=list(six_cluster.loc['limit
_usage',:].values)
cash_advance
```

Out[172]:
```
array([5.219, 3.027, 1.282, 1.393, 5.32
5, 5.492])
```

Conclusion with 6 clusters:

- Here also groups are overlapping.
  - Cl-0 and Cl-2 behaving same

Checking performance metrics for Kmeans

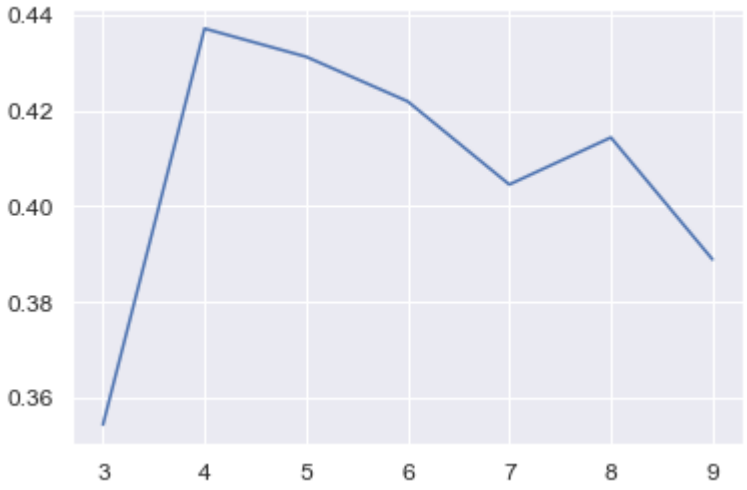
- I am validating performance with 2 metrics Calinski harabaz and Silhouette score

In [174]:
```
from sklearn.metrics import calinski_harabaz_score,silhouette_score
```

In [175]:
```
score={}
score_c={}
for n in range(3,10):
    km_score=KMeans(n_clusters=n)
    km_score.fit(reduced_cr)
    score_c[n]=calinski_harabaz_score(reduced_cr,km_score.labels_)
    score[n]=silhouette_score(reduced_cr,km_score.labels_)
```

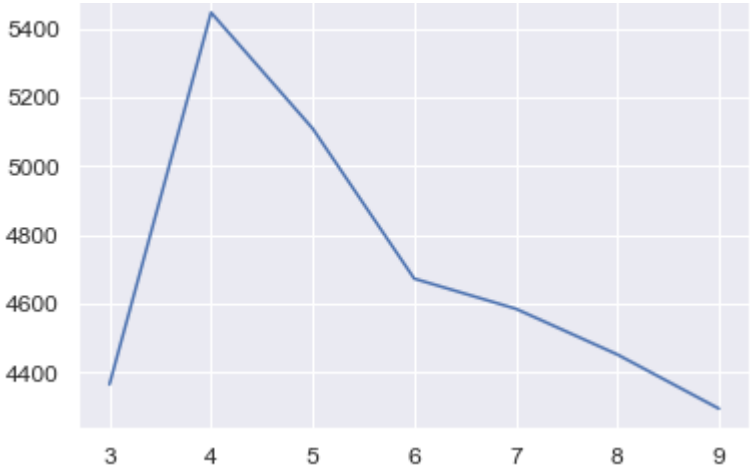
In [176]:
```
pd.Series(score).plot()
```

Out[176]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1333fcc0>
```



In [177]:
```
pd.Series(score_c).plot()
```

Out[177]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x133ad668>
```



Performance metrics also suggest that K-means with 4 cluster is able to show distinguished characteristics of each cluster.

Insights with 4 Clusters

- Cluster 2 is the group of customers who have highest Monthly_avg purchases and doing both installment as well as one_off purchases, have comparatively good credit score. ***This group is about 31% of the total customer base***
- cluster 1 is taking maximum advance_cash and is paying comparatively less minimum payment and poor credit_score & doing no purchase transaction. ***This group is about 23% of the total customer base***
- Cluster 0 customers are doing maximum One_Off transactions and least payment ratio and credit_score on lower side ***This group is about 21% of the total customer base***
- Cluster 3 customers have maximum credit score and are paying dues and are doing maximum installment purchases. ***This group is about 25% of the total customer base***

Marketing Strategy Suggested:

a. Group 2

- They are potential target customers who are paying dues and doing purchases and maintaining comparatively good credit score) -- we can increase credit limit or can lower

down interest rate -- Can be given premium card /loyality cards to increase transactions

Nov 10, 2019 (http://www.datasciencelovers.com/2019/11/)

Network Intrusion Detection (http://www.datasciencelovers.com/machine-learning-projects/network-intrusion-detection/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/network-intrusion-detection/) Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/)** Tag **classification (http://www.datasciencelovers.com/tag/classification/), logistic regression (http://www.datasciencelovers.com/tag/logistic-regression/), machine learning (http://www.datasciencelovers.com/tag/machine-learning/), multi class classification (http://www.datasciencelovers.com/tag/multi-class-classification/), Network intrusion detection (http://www.datasciencelovers.com/tag/network-intrusion-detection/)**

In this case study we need to predict anomalies and attacks in the network.

Business Problem:

The task is to build network intrusion detection system to detect anomalies and attacks in the network.

There are two problems.

1. **Binomial** Classification: Activity is normal or attack.
2. **Multinomial** classification: Activity is normal or DOS or PROBE or R2L or U2R .

Data Availability:

This data is KDDCUP'99 data set, which is widely used as one of the few publicly available data sets for network-based anomaly detection systems.

For more about data you can visit to http://www.unb.ca/cic/datasets/nsl.html (http://www.unb.ca/cic/datasets/nsl.html)

BASIC FEATURES OF EACH NETWORK CONNECTION VECTOR

1. **Duration:** Length of time duration of the connection
2. **Protocol_type:** Protocol used in the connection
3. **Service:** Destination network service used
4. **Flag:** Status of the connection – Normal or Error
5. **Src_bytes:** Number of data bytes transferred from source to destination in single connection
6. **Dst_bytes:** Number of data bytes transferred from destination to source in single connection

7. **Land:** if source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0
8. **Wrong_fragment:** Total number of wrong fragments in this connection
9. **Urgent:** Number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated.
10. **Hot:** Number of „hot" indicators in the content such as: entering a system directory, creating programs and executing programs.
11. **Num_failed _logins:** Count of failed login attempts.
12. **Logged_in Login Status**: 1 if successfully logged in; 0 otherwise.
13. **Num_compromised**: Number of "compromised' ' conditions.
14. **Root_shell:** 1 if root shell is obtained; 0 otherwise.
15. **Su_attempted:** 1 if "su root" command attempted or used; 0 otherwise.
16. **Num_root:** Number of "root" accesses or number of operations performed as a root in the connection.
17. **Num_file_creations:** Number of file creation operations in the connection.
18. **Num_shells:** Number of shell prompts.
19. **Num_access_files**: Number of operations on access control files .
20. **Num_outbound_cmds:** Number of outbound commands in an ftp session.
21. **Is_hot_login:** 1 if the login belongs to the "hot" list i.e., root or admin; else 0.
22. **Is_guest_login:** 1 if the login is a "guest" login; 0 otherwise .
23. **Count:** Number of connections to the same destination host as the current connection in the past two seconds
24. **Srv_count:** Number of connections to the same service (port number) as the current connection in the past two seconds.
25. **Serror_rate:** The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23)
26. **Srv_serror_rate:** The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24)
27. **Rerror_rate:** The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23)
28. **Srv_rerror_rate:** The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24)
29. **Same_srv_rate:** The percentage of connections that were to the same service, among the connections aggregated in count (23)
30. **Diff_srv_rate:** The percentage of connections that were to different services, among the connections aggregated in count (23)
31. **Srv_diff_host_ rate:** The percentage of connections that were to different destination machines among the connections aggregated in srv_count (24)
32. **Dst_host_count:** Number of connections having the same destination host IP address.
33. **Dst_host_srv_ count:** Number of connections having the same port number.

34. **Dst_host_same _srv_rate:** The percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32) .
35. **Dst_host_diff_ srv_rate:** The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32)
36. **Dst_host_same _src_port_rate:** The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_c ount (33) .
37. **Dst_host_srv_ diff_host_rate:** The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (33).
38. **Dst_host_serro r_rate:** The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (32).
39. **Dst_host_srv_s error_rate:** The percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_c ount (33).
40. **Dst_host_rerro r_rate:** The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32) .
41. **Dst_host_srv_r error_rate:** The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_c ount (33).

Attack Class:

Attack Class	Attack Type
DoS	Back, Land, Neptune, Pod, Smurf,Teardrop,Apache2, Udpstorm, Processtable, Worm (10)
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint (6)
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named (16)
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps (7)

Let’s develop a machine learning model for further analysis.

NETWORK INTRUSION DETECTION WITH MACHINE LEARNING.

Solution approach description

Step 1: Data preprocessing:

All features are made numerical using one-Hot-encoding. The features are scaled to avoid features with large values that may weigh too much in the results.

Step 2: Feature Selection:

Eliminate redundant and irrelevant data by selecting a subset of relevant features that fully represents the given problem. Univariate feature selection with ANOVA F-test. This analyzes each feature individually to detemine the strength of the relationship between the feature and labels. Using SecondPercentile method (sklearn.feature_selection) to select features based on percentile of the highest scores. When this subset is found: Recursive Feature Elimination (RFE) is applied.

Step 4: Build the model:

Decision tree model is built.

Step 5: Prediction & Evaluation (validation):

Using the test data to make predictions of the model. Multiple scores are considered such as:accuracy score, recall, f-measure, confusion matrix. perform a 10-fold cross-validation

Machine learning work steps through python:

Import Library:

In [1]:
```
# import relevant modules
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as sn
import scipy.stats as stats
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
from sklearn.linear_model import LogisticRegression
from statsmodels.stats.outliers_influence import variance_inflat
ion_factor
from patsy import dmatrices
%matplotlib inline

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Settings
pd.set_option('display.max_columns', None)
np.set_printoptions(threshold=np.nan)
np.set_printoptions(precision=3)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

Load the data set

In [8]:
```
# As column name is not defined in the data set need to define c
olumn names to the data set
col_names = ["duration","protocol_type","service","flag","src_by
tes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_fail
ed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","n
um_root",
    "num_file_creations","num_shells","num_access_files","num_ou
tbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","serror
_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_
rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_h
ost_srv_count",
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_
same_src_port_rate",
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_ho
st_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","label",
"last_flag"]
```

In [9]:
```
# import train data set
df = pd.read_table("Train.txt", sep=",", names=col_names)
df = df.iloc[:,:-1] # removes an unwanted extra field
```

In [11]:
```
# import test data set
df_test = pd.read_table("Test.txt", sep=",", names=col_names)
df_test = df_test.iloc[:,:-1]
```

In [12]:
```
# shape, this gives the dimensions of the dataset
print('Dimensions of the Training set:',df.shape)
print('Dimensions of the Test set:',df_test.shape)
```

```
Dimensions of the Training set: (125973, 42)
Dimensions of the Test set: (22544, 42)
```

Sample view of the training dataset:

In [13]:
```
# first five rows of train data set
df.head(5)
```

Out[13]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fra
0	0	tcp	ftp_data	SF	491	0	0	
1	0	udp	other	SF	146	0	0	
2	0	tcp	private	S0	0	0	0	
3	0	tcp	http	SF	232	8153	0	
4	0	tcp	http	SF	199	420	0	

In [14]:
```
# first five rows of test data set
df_test.head(5)
```

Out[14]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_f
0	0	tcp	private	REJ	0	0	0	
1	0	tcp	private	REJ	0	0	0	
2	2	tcp	ftp_data	SF	12983	0	0	
3	0	icmp	eco_i	SF	20	0	0	
4	1	tcp	telnet	RSTO	0	15	0	

Statistical Summary

In [18]:
```
df.describe()
```

Out[18]:

	duration	src_bytes	dst_bytes	land	wrong_fragme
count	125973.00000	1.259730e+05	1.259730e+05	125973.000000	125973.00000
mean	287.14465	4.556674e+04	1.977911e+04	0.000198	0.02268
std	2604.51531	5.870331e+06	4.021269e+06	0.014086	0.25353
min	0.00000	0.000000e+00	0.000000e+00	0.000000	0.00000
25%	0.00000	0.000000e+00	0.000000e+00	0.000000	0.00000
50%	0.00000	4.400000e+01	0.000000e+00	0.000000	0.00000
75%	0.00000	2.760000e+02	5.160000e+02	0.000000	0.00000
max	42908.00000	1.379964e+09	1.309937e+09	1.000000	3.00000

Label Distribution of Training and Test set

Label distribution of Training and Test set

In [19]:
```
print('Label distribution Training set:')
print(df['label'].value_counts())
print()
print('Label distribution Test set:')
print(df_test['label'].value_counts())
```

```
Label distribution Training set:
normal              67343
neptune             41214
satan                3633
ipsweep              3599
portsweep            2931
smurf                2646
nmap                 1493
back                  956
teardrop              892
warezclient           890
pod                   201
guess_passwd           53
buffer_overflow        30
warezmaster            20
land                   18
imap                   11
rootkit                10
loadmodule              9
ftp_write               8
multihop                7
phf                     4
perl                    3
spy                     2
Name: label, dtype: int64

Label distribution Test set:
normal              9711
neptune             4657
guess_passwd        1231
mscan                996
warezmaster          944
apache2              737
satan                735
processtable         685
smurf                665
back                 359
snmpguess            331
saint                319
mailbomb             293
snmpgetattack        178
portsweep            157
ipsweep              141
httptunnel           133
nmap                  73
pod                   41
buffer_overflow       20
multihop              18
named                 17
ps                    15
sendmail              14
xterm                 13
rootkit               13
teardrop              12
```

```
xlock                9
land                 7
xsnoop               4
ftp_write            3
worm                 2
udpstorm             2
perl                 2
sqlattack            2
loadmodule           2
phf                  2
imap                 1
Name: label, dtype: int64
```

Step 1: Data preprocessing:

One-Hot-Encoding (one-of-K) is used to to transform all categorical features into binary features. Requirement for One-Hot-encoding: "The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete) features. The output will be a sparse matrix where each column corresponds to one possible value of one feature. It is assumed that input features take on values in the range (0, n_values)."

Therefore the features first need to be transformed with LabelEncoder, to transform every category to a number.

Identify categorical features

In [20]:
```
# colums that are categorical and not binary yet: protocol_type
(column 2), service (column 3), flag (column 4).
# explore categorical features
print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categorie
s".format(col_name=col_name, unique_cat=unique_cat))

#see how distributed the feature service is, it is evenly distri
buted and therefore we need to make dummies for all.
print()
print('Distribution of categories in service:')
print(df['service'].value_counts().sort_values(ascending=False).
head())
```

```
Training set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 70 categories
Feature 'flag' has 11 categories
Feature 'label' has 23 categories

Distribution of categories in service:
http        40338
private     21853
domain_u     9043
smtp         7313
ftp_data     6860
Name: service, dtype: int64
```

In [21]:
```
# Same loop we will apply on Test set and find the categorical features.
print('Test set:')
for col_name in df_test.columns:
    if df_test[col_name].dtypes == 'object' :
        unique_cat = len(df_test[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))
```

```
Test set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 64 categories
Feature 'flag' has 11 categories
Feature 'label' has 38 categories
```

Conclusion: Need to make dummies for all categories as the distribution is fairly even. In total: 3+70+11=84 dummies.

Comparing the results shows that the Test set has fewer categories (6), these need to be added as empty columns.

LabelEncoder

Insert categorical features into a 2D numpy array

In [22]:
```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder


# insert code to get a list of categorical columns into a variable, categorical_columns
categorical_columns=['protocol_type', 'service', 'flag']

# Get the categorical values into a 2D numpy array
df_categorical_values = df[categorical_columns]

testdf_categorical_values = df_test[categorical_columns]
df_categorical_values.head()
```

Out[22]:

	protocol_type	service	flag
0	tcp	ftp_data	SF
1	udp	other	SF
2	tcp	private	S0
3	tcp	http	SF
4	tcp	http	SF

Make column names for dummies

In [23]:
```
# For protocol type
unique_protocol=sorted(df.protocol_type.unique())
string1 = 'Protocol_type_'
```

```
unique_protocol2=[string1 + x for x in unique_protocol]

# For service
unique_service=sorted(df.service.unique())
string2 = 'service_'
unique_service2=[string2 + x for x in unique_service]

# For flag
unique_flag=sorted(df.flag.unique())
string3 = 'flag_'
unique_flag2=[string3 + x for x in unique_flag]

# put together
dumcols=unique_protocol2 + unique_service2 + unique_flag2
print(dumcols)

#do same for test set
unique_service_test=sorted(df_test.service.unique())
unique_service2_test=[string2 + x for x in unique_service_test]
testdumcols=unique_protocol2 + unique_service2_test + unique_fla
g2
```

```
['Protocol_type_icmp', 'Protocol_type_tcp', 'Protocol_type_ud
p', 'service_IRC', 'service_X11', 'service_Z39_50', 'service_ao
l', 'service_auth', 'service_bgp', 'service_courier', 'service_
csnet_ns', 'service_ctf', 'service_daytime', 'service_discard',
'service_domain', 'service_domain_u', 'service_echo', 'service_
eco_i', 'service_ecr_i', 'service_efs', 'service_exec', 'servic
e_finger', 'service_ftp', 'service_ftp_data', 'service_gopher',
'service_harvest', 'service_hostnames', 'service_http', 'servic
e_http_2784', 'service_http_443', 'service_http_8001', 'service
_imap4', 'service_iso_tsap', 'service_klogin', 'service_kshel
l', 'service_ldap', 'service_link', 'service_login', 'service_m
tp', 'service_name', 'service_netbios_dgm', 'service_netbios_n
s', 'service_netbios_ssn', 'service_netstat', 'service_nnsp',
'service_nntp', 'service_ntp_u', 'service_other', 'service_pm_d
ump', 'service_pop_2', 'service_pop_3', 'service_printer', 'ser
vice_private', 'service_red_i', 'service_remote_job', 'service_
rje', 'service_shell', 'service_smtp', 'service_sql_net', 'serv
ice_ssh', 'service_sunrpc', 'service_supdup', 'service_systat',
'service_telnet', 'service_tftp_u', 'service_tim_i', 'service_t
ime', 'service_urh_i', 'service_urp_i', 'service_uucp', 'servic
e_uucp_path', 'service_vmnet', 'service_whois', 'flag_OTH', 'fl
ag_REJ', 'flag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0', 'f
lag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH']
```

Categorical features transform into numbers with the help of LabelEncoder()

In [24]:
```
df_categorical_values_enc=df_categorical_values.apply(LabelEncod
er().fit_transform)
print(df_categorical_values_enc.head())

# test set
testdf_categorical_values_enc=testdf_categorical_values.apply(La
belEncoder().fit_transform)
```

```
   protocol_type  service  flag
0              1       20     9
1              2       44     9
2              1       49     5
```

3	1	24	9
4	1	24	9

Apply One-Hot-Encoding

In [25]:
```
enc = OneHotEncoder()
df_categorical_values_encenc = enc.fit_transform(df_categorical_values_enc)
df_cat_data = pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)

# test set
testdf_categorical_values_encenc = enc.fit_transform(testdf_categorical_values_enc)
testdf_cat_data = pd.DataFrame(testdf_categorical_values_encenc.toarray(),columns=testdumcols)

df_cat_data.head()
```

Out[25]:

	Protocol_type_icmp	Protocol_type_tcp	Protocol_type_udp	service_IRC	servi
0	0.0	1.0	0.0	0.0	
1	0.0	0.0	1.0	0.0	
2	0.0	1.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	
4	0.0	1.0	0.0	0.0	

Add 6 missing categories in service from train set to test set.

In [26]:
```
trainservice=df['service'].tolist()
testservice= df_test['service'].tolist()
difference=list(set(trainservice) - set(testservice))
string = 'service_'
difference=[string + x for x in difference]
difference
```

Out[26]:
```
['service_http_8001',
 'service_urh_i',
 'service_red_i',
 'service_http_2784',
 'service_harvest',
 'service_aol']
```

In [27]:
```
for col in difference:
    testdf_cat_data[col] = 0

testdf_cat_data.shape
```

Out[27]: (22544, 84)

Join encoded categorical dataframe with the non-categorical dataframe

Join encoded categorical dataframe with the non-categorical dataframe

In [28]:
```
newdf=df.join(df_cat_data)
newdf.drop('flag', axis=1, inplace=True)
newdf.drop('protocol_type', axis=1, inplace=True)
newdf.drop('service', axis=1, inplace=True)

# test data
newdf_test=df_test.join(testdf_cat_data)
newdf_test.drop('flag', axis=1, inplace=True)
newdf_test.drop('protocol_type', axis=1, inplace=True)
newdf_test.drop('service', axis=1, inplace=True)
print(newdf.shape)
print(newdf_test.shape)
```

```
(125973, 123)
(22544, 123)
```

Split Dataset into 4 datasets for every attack category

Rename every attack label: 0=normal, 1=DoS, 2=Probe, 3=R2L and 4=U2R.

Replace labels column with new labels columng

Make new datasets

In [29]:
```
# take label column
labeldf=newdf['label']
labeldf_test=newdf_test['label']

# change the label column
newlabeldf=labeldf.replace({ 'normal' : 0, 'neptune' : 1 ,'bac
k': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'mailbom
b': 1, 'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'worm':
1,
                            'ipsweep' : 2,'nmap' : 2,'portsweep'
: 2,'satan' : 2,'mscan' : 2,'saint' : 2
                            ,'ftp_write': 3,'guess_passwd': 3,'im
ap': 3,'multihop': 3,'phf': 3,'spy': 3,'warezclient': 3,'warezma
ster': 3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpgues
s': 3,'xlock': 3,'xsnoop': 3,'httptunnel': 3,
                            'buffer_overflow': 4,'loadmodule':
4,'perl': 4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})

newlabeldf_test=labeldf_test.replace({ 'normal' : 0, 'neptune' :
1 ,'back': 1, 'land': 1, 'pod': 1, 'smurf': 1, 'teardrop': 1,'ma
ilbomb': 1, 'apache2': 1, 'processtable': 1, 'udpstorm': 1, 'wor
m': 1,
                            'ipsweep' : 2,'nmap' : 2,'portsweep'
: 2,'satan' : 2,'mscan' : 2,'saint' : 2
                            ,'ftp_write': 3,'guess_passwd': 3,'im
ap': 3,'multihop': 3,'phf': 3,'spy': 3,'warezclient': 3,'warezma
ster': 3,'sendmail': 3,'named': 3,'snmpgetattack': 3,'snmpgues
s': 3,'xlock': 3,'xsnoop': 3,'httptunnel': 3,
                            'buffer_overflow': 4,'loadmodule':
4,'perl': 4,'rootkit': 4,'ps': 4,'sqlattack': 4,'xterm': 4})

# put the new label column back
```

```
newdf['label'] = newlabeldf
newdf_test['label'] = newlabeldf_test

print(newdf['label'].head())
```

```
0    0
1    0
2    1
3    0
4    0
Name: label, dtype: int64
```

In [30]:
```
to_drop_DoS = [2,3,4]
to_drop_Probe = [1,3,4]
to_drop_R2L = [1,2,4]
to_drop_U2R = [1,2,3]

# create dtafame accorrding to level dype for trining data
DoS_df=newdf[~newdf['label'].isin(to_drop_DoS)];
Probe_df=newdf[~newdf['label'].isin(to_drop_Probe)];
R2L_df=newdf[~newdf['label'].isin(to_drop_R2L)];
U2R_df=newdf[~newdf['label'].isin(to_drop_U2R)];

#create dtafame accorrding to level type for testing data
DoS_df_test=newdf_test[~newdf_test['label'].isin(to_drop_DoS)];
Probe_df_test=newdf_test[~newdf_test['label'].isin(to_drop_Prob
e)];
R2L_df_test=newdf_test[~newdf_test['label'].isin(to_drop_R2L)];
U2R_df_test=newdf_test[~newdf_test['label'].isin(to_drop_U2R)];

print('Train:')
print('Dimensions of DoS:' ,DoS_df.shape)
print('Dimensions of Probe:' ,Probe_df.shape)
print('Dimensions of R2L:' ,R2L_df.shape)
print('Dimensions of U2R:' ,U2R_df.shape)
print('Test:')
print('Dimensions of DoS:' ,DoS_df_test.shape)
print('Dimensions of Probe:' ,Probe_df_test.shape)
print('Dimensions of R2L:' ,R2L_df_test.shape)
print('Dimensions of U2R:' ,U2R_df_test.shape)
```

```
Train:
Dimensions of DoS: (113270, 123)
Dimensions of Probe: (78999, 123)
Dimensions of R2L: (68338, 123)
Dimensions of U2R: (67395, 123)
Test:
Dimensions of DoS: (17171, 123)
Dimensions of Probe: (12132, 123)
Dimensions of R2L: (12596, 123)
Dimensions of U2R: (9778, 123)
```

Step 2: Feature Scaling:

In [31]:
```
# Split dataframes into X & Y
# assign X as a dataframe of feautures and Y as a series of outc
ome variables
X_DoS = DoS_df.drop('label',1)
Y_DoS = DoS_df.label
```

```
X_Probe = Probe_df.drop('label',1)
Y_Probe = Probe_df.label
X_R2L = R2L_df.drop('label',1)
Y_R2L = R2L_df.label
X_U2R = U2R_df.drop('label',1)
Y_U2R = U2R_df.label

# test set
X_DoS_test = DoS_df_test.drop('label',1)
Y_DoS_test = DoS_df_test.label
X_Probe_test = Probe_df_test.drop('label',1)
Y_Probe_test = Probe_df_test.label
X_R2L_test = R2L_df_test.drop('label',1)
Y_R2L_test = R2L_df_test.label
X_U2R_test = U2R_df_test.drop('label',1)
Y_U2R_test = U2R_df_test.label
```

Save a list of feature names for later use (it is the same for every attack category). Column names are dropped at this stage.

In [32]:
```
colNames=list(X_DoS)
colNames_test=list(X_DoS_test)
```

In [33]:
```
from sklearn import preprocessing

# For train data set
scaler1 = preprocessing.StandardScaler().fit(X_DoS)
X_DoS=scaler1.transform(X_DoS)
scaler2 = preprocessing.StandardScaler().fit(X_Probe)
X_Probe=scaler2.transform(X_Probe)
scaler3 = preprocessing.StandardScaler().fit(X_R2L)
X_R2L=scaler3.transform(X_R2L)
scaler4 = preprocessing.StandardScaler().fit(X_U2R)
X_U2R=scaler4.transform(X_U2R)

# For test data
scaler5 = preprocessing.StandardScaler().fit(X_DoS_test)
X_DoS_test=scaler5.transform(X_DoS_test)
scaler6 = preprocessing.StandardScaler().fit(X_Probe_test)
X_Probe_test=scaler6.transform(X_Probe_test)
scaler7 = preprocessing.StandardScaler().fit(X_R2L_test)
X_R2L_test=scaler7.transform(X_R2L_test)
scaler8 = preprocessing.StandardScaler().fit(X_U2R_test)
X_U2R_test=scaler8.transform(X_U2R_test)
```

In [34]:
```
print(X_DoS.std(axis=0))
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1.
1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0.
1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
1. 1. 1.
 1. 1.]
```

In [35]:
```
X_Probe.std(axis=0);
X_R2L.std(axis=0);
X_U2R.std(axis=0);
```

Step 3: Feature Selection:

1. Univariate Feature Selection using ANOVA F-test

In [36]:
```
#univariate feature selection with ANOVA F-test. using secondPer
centile method, then RFE
#Scikit-learn exposes feature selection routines as objects that
implement the transform method
#SelectPercentile: removes all but a user-specified highest scor
ing percentage of features
#f_classif: ANOVA F-value between label/feature for classificati
on tasks.
from sklearn.feature_selection import SelectPercentile, f_classi
f
np.seterr(divide='ignore', invalid='ignore');
selector=SelectPercentile(f_classif, percentile=10)
X_newDoS = selector.fit_transform(X_DoS,Y_DoS)
X_newDoS.shape
```

Out[36]:
```
(113270, 13)
```

Get the features that were selected for DoS

In [37]:
```
true=selector.get_support()
newcolindex_DoS=[i for i, x in enumerate(true) if x]
newcolname_DoS=list( colNames[i] for i in newcolindex_DoS )
newcolname_DoS
```

Out[37]:
```
['logged_in',
 'count',
 'serror_rate',
 'srv_serror_rate',
 'same_srv_rate',
 'dst_host_count',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_serror_rate',
 'dst_host_srv_serror_rate',
 'service_http',
 'flag_S0',
 'flag_SF']
```

In [38]:
```
X_newProbe = selector.fit_transform(X_Probe,Y_Probe)
X_newProbe.shape
```

Out[38]:
```
(78999, 13)
```

Get the features that were selected: Probe

In [39]:
```
true=selector.get_support()
newcolindex_Probe=[i for i, x in enumerate(true) if x]
newcolname_Probe=list( colNames[i] for i in newcolindex_Probe )
newcolname_Probe
```

Out[39]:
```
['logged_in',
 'rerror_rate',
 'srv_rerror_rate',
 'dst_host_srv_count',
 'dst_host_diff_srv_rate',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'dst_host_rerror_rate',
 'dst_host_srv_rerror_rate',
 'Protocol_type_icmp',
 'service_eco_i',
 'service_private',
 'flag_SF']
```

In [40]:
```
X_newR2L = selector.fit_transform(X_R2L,Y_R2L)
X_newR2L.shape
```

Out[40]:
```
(68338, 13)
```

Get the features that were selected: R2L

In [41]:
```
true=selector.get_support()
newcolindex_R2L=[i for i, x in enumerate(true) if x]
newcolname_R2L=list( colNames[i] for i in newcolindex_R2L)
newcolname_R2L
```

Out[41]:
```
['src_bytes',
 'dst_bytes',
 'hot',
 'num_failed_logins',
 'is_guest_login',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'service_ftp',
 'service_ftp_data',
 'service_http',
 'service_imap4',
 'flag_RSTO']
```

In [42]:
```
X_newU2R = selector.fit_transform(X_U2R,Y_U2R)
X_newU2R.shape
```

Out[42]:
```
(67395, 13)
```

Get the features that were selected: U2R

In [43]:
```
true=selector.get_support()
newcolindex_U2R=[i for i, x in enumerate(true) if x]
newcolname_U2R=list( colNames[i] for i in newcolindex_U2R)
newcolname_U2R
```

Out[43]:
```
['urgent',
 'hot',
 'root_shell',
 'num_file_creations',
 'num_shells',
 'srv_diff_host_rate',
 'dst_host_count',
 'dst_host_srv_count',
 'dst_host_same_src_port_rate',
 'dst_host_srv_diff_host_rate',
 'service_ftp_data',
 'service_http',
 'service_telnet']
```

Summary of features selected by Univariate Feature Selection

In [44]:
```
print('Features selected for DoS:',newcolname_DoS)
print()
print('Features selected for Probe:',newcolname_Probe)
print()
print('Features selected for R2L:',newcolname_R2L)
print()
print('Features selected for U2R:',newcolname_U2R)
```

```
Features selected for DoS: ['logged_in', 'count', 'serror_rat
e', 'srv_serror_rate', 'same_srv_rate', 'dst_host_count', 'dst_
host_srv_count', 'dst_host_same_srv_rate', 'dst_host_serror_rat
e', 'dst_host_srv_serror_rate', 'service_http', 'flag_S0', 'fla
g_SF']

Features selected for Probe: ['logged_in', 'rerror_rate', 'srv_
rerror_rate', 'dst_host_srv_count', 'dst_host_diff_srv_rate',
'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'Protocol_t
ype_icmp', 'service_eco_i', 'service_private', 'flag_SF']

Features selected for R2L: ['src_bytes', 'dst_bytes', 'hot', 'n
um_failed_logins', 'is_guest_login', 'dst_host_srv_count', 'dst
_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'serv
ice_ftp', 'service_ftp_data', 'service_http', 'service_imap4',
'flag_RSTO']

Features selected for U2R: ['urgent', 'hot', 'root_shell', 'num
_file_creations', 'num_shells', 'srv_diff_host_rate', 'dst_host
_count', 'dst_host_srv_count', 'dst_host_same_src_port_rate',
'dst_host_srv_diff_host_rate', 'service_ftp_data', 'service_htt
p', 'service_telnet']
```

After obtaining the adequate number of features during the univariate selection process, a recursive feature elimination (RFE) was operated with the number of features passed as parameter to identify the features selected". This either implies that RFE is only used for obtaining the features previously selected but also obtaining the rank. This use of RFE is however very redundant as the features selected can be obtained in another way (Done in this project). One can also not say that the features were selected by RFE, as it was not used for this. The quote could however also imply that only the number 13 from univariate feature selection was used. RFE is then used for feature selection trying to

find the best 13 features. With this use of RFE one can actually say that it was used for feature selection.

To procede with the data mining, the second option is considered as this uses RFE. From now on the number of features for every attack category is 13.

2. Recursive Feature Elimination for feature ranking

Option 1: get importance from previous selected)

In [46]:
```
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
# Create a decision tree classifier. By convention, clf means 'c
lassifier'
clf = DecisionTreeClassifier(random_state=0)

#rank all features, i.e continue the elimination until the last
one
rfe = RFE(clf, n_features_to_select=1)
rfe.fit(X_newDoS, Y_DoS)
print ("DoS Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newc
olname_DoS)))
```

```
DoS Features sorted by their rank:
[(1, 'same_srv_rate'), (2, 'count'), (3, 'flag_SF'), (4, 'dst_h
ost_serror_rate'), (5, 'dst_host_same_srv_rate'), (6, 'dst_host
_srv_count'), (7, 'dst_host_count'), (8, 'logged_in'), (9, 'ser
ror_rate'), (10, 'dst_host_srv_serror_rate'), (11, 'srv_serror_
rate'), (12, 'service_http'), (13, 'flag_S0')]
```

In [47]:
```
rfe.fit(X_newProbe, Y_Probe)
print ("Probe Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newc
olname_Probe)))
```

```
Probe Features sorted by their rank:
[(1, 'dst_host_same_src_port_rate'), (2, 'dst_host_srv_count'),
(3, 'dst_host_rerror_rate'), (4, 'service_private'), (5, 'logge
d_in'), (6, 'dst_host_diff_srv_rate'), (7, 'dst_host_srv_diff_h
ost_rate'), (8, 'flag_SF'), (9, 'service_eco_i'), (10, 'rerror_
rate'), (11, 'Protocol_type_icmp'), (12, 'dst_host_srv_rerror_r
ate'), (13, 'srv_rerror_rate')]
```

In [48]:
```
rfe.fit(X_newR2L, Y_R2L)

print ("R2L Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newc
olname_R2L)))
```

```
R2L Features sorted by their rank:
[(1, 'src_bytes'), (2, 'dst_bytes'), (3, 'hot'), (4, 'dst_host_
srv_diff_host_rate'), (5, 'service_ftp_data'), (6, 'dst_host_sa
me_src_port_rate'), (7, 'dst_host_srv_count'), (8, 'num_failed_
logins'), (9, 'service_imap4'), (10, 'is_guest_login'), (11, 's
ervice_ftp'), (12, 'flag_RSTO'), (13, 'service_http')]
```

In [49]:
```
rfe.fit(X_newU2R, Y_U2R)

print ("U2R Features sorted by their rank:")
print (sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), newc
olname_U2R)))
```

```
U2R Features sorted by their rank:
[(1, 'hot'), (2, 'dst_host_srv_count'), (3, 'dst_host_count'),
(4, 'root_shell'), (5, 'num_shells'), (6, 'service_ftp_data'),
(7, 'dst_host_srv_diff_host_rate'), (8, 'num_file_creations'),
(9, 'dst_host_same_src_port_rate'), (10, 'service_telnet'), (1
1, 'srv_diff_host_rate'), (12, 'service_http'), (13, 'urgent')]
```

Option 2: get 13 best features from 122 from RFE)

In [50]:
```
from sklearn.feature_selection import RFE
clf = DecisionTreeClassifier(random_state=0)
rfe = RFE(estimator=clf, n_features_to_select=13, step=1)
rfe.fit(X_DoS, Y_DoS)
X_rfeDoS=rfe.transform(X_DoS)
true=rfe.support_
rfecolindex_DoS=[i for i, x in enumerate(true) if x]
rfecolname_DoS=list(colNames[i] for i in rfecolindex_DoS)
```

In [51]:
```
rfe.fit(X_Probe, Y_Probe)
X_rfeProbe=rfe.transform(X_Probe)
true=rfe.support_
rfecolindex_Probe=[i for i, x in enumerate(true) if x]
rfecolname_Probe=list(colNames[i] for i in rfecolindex_Probe)
```

In [52]:
```
rfe.fit(X_R2L, Y_R2L)
X_rfeR2L=rfe.transform(X_R2L)
true=rfe.support_
rfecolindex_R2L=[i for i, x in enumerate(true) if x]
rfecolname_R2L=list(colNames[i] for i in rfecolindex_R2L)
```

In [53]:
```
rfe.fit(X_U2R, Y_U2R)
X_rfeU2R=rfe.transform(X_U2R)
true=rfe.support_
rfecolindex_U2R=[i for i, x in enumerate(true) if x]
rfecolname_U2R=list(colNames[i] for i in rfecolindex_U2R)
```

Summary of features selected by RFE

In [54]:
```
print('Features selected for DoS:',rfecolname_DoS)
print()
print('Features selected for Probe:',rfecolname_Probe)
print()
print('Features selected for R2L:',rfecolname_R2L)
print()
print('Features selected for U2R:',rfecolname_U2R)
```

```
Features selected for DoS: ['src_bytes', 'dst_bytes', 'wrong_fr
agment', 'num_compromised', 'same_srv_rate', 'diff_srv_rate',
'dst_host_count', 'dst_host_same_srv_rate', 'dst_host_serror_ra
```

```
'dst_host_count', 'dst_host_same_srv_rate', 'dst_host_serror_ra
te', 'dst_host_srv_serror_rate', 'service_ecr_i', 'flag_RSTR',
'flag_S0']

Features selected for Probe: ['src_bytes', 'dst_bytes', 'rerror
_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'ds
t_host_same_src_port_rate', 'dst_host_rerror_rate', 'service_fi
nger', 'service_ftp_data', 'service_http', 'service_private',
'service_smtp', 'service_telnet']

Features selected for R2L: ['duration', 'src_bytes', 'dst_byte
s', 'hot', 'num_failed_logins', 'num_access_files', 'dst_host_c
ount', 'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_hos
t_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'service_
ftp_data', 'service_imap4']

Features selected for U2R: ['duration', 'src_bytes', 'dst_byte
s', 'hot', 'root_shell', 'num_file_creations', 'num_shells', 's
rv_count', 'dst_host_count', 'dst_host_same_srv_rate', 'dst_hos
t_srv_diff_host_rate', 'service_ftp_data', 'service_other']
```

In [55]:
```
#Size of each data frame
print(X_rfeDoS.shape)
print(X_rfeProbe.shape)
print(X_rfeR2L.shape)
print(X_rfeU2R.shape)
```

```
(113270, 13)
(78999, 13)
(68338, 13)
(67395, 13)
```

Build the model

Classifier is trained for all features and for reduced features, for later comparison.

In [56]:
```
# For all features
clf_DoS=DecisionTreeClassifier(random_state=0)
clf_Probe=DecisionTreeClassifier(random_state=0)
clf_R2L=DecisionTreeClassifier(random_state=0)
clf_U2R=DecisionTreeClassifier(random_state=0)
clf_DoS.fit(X_DoS, Y_DoS)
clf_Probe.fit(X_Probe, Y_Probe)
clf_R2L.fit(X_R2L, Y_R2L)
clf_U2R.fit(X_U2R, Y_U2R)
```

Out[56]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max
_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random
_state=0,
            splitter='best')
```

In [57]:
```
# For selected features
clf_rfeDoS=DecisionTreeClassifier(random_state=0)
```

```
clf_rfeProbe=DecisionTreeClassifier(random_state=0)
clf_rfeR2L=DecisionTreeClassifier(random_state=0)
clf_rfeU2R=DecisionTreeClassifier(random_state=0)
clf_rfeDoS.fit(X_rfeDoS, Y_DoS)
clf_rfeProbe.fit(X_rfeProbe, Y_Probe)
clf_rfeR2L.fit(X_rfeR2L, Y_R2L)
clf_rfeU2R.fit(X_rfeU2R, Y_U2R)
```

Out[57]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max
_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random
_state=0,
            splitter='best')
```

Prediction & Evaluation (validation):

Using all Features for each category

Confusion Matrices

In [ ]:
```
# Apply the classifier we trained to the test data (which it has
never seen before)
clf_DoS.predict(X_DoS_test)
```

In [59]:
```
# View the predicted probabilities of the first 10 observations
clf_DoS.predict_proba(X_DoS_test)[0:10]
```

Out[59]:
```
array([[0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.]])
```

In [60]:
```
Y_DoS_pred=clf_DoS.predict(X_DoS_test)
# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred, rownames=['Actual attacks'],
colnames=['Predicted attacks'])
```

Out[60]:

Predicted attacks	0	1
Actual attacks		
0	9499	212
1	2830	4630

For Probe:

In [61]:
```
Y_Probe_pred=clf_Probe.predict(X_Probe_test)

# Create confusion matrix
pd.crosstab(Y_Probe_test, Y_Probe_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[61]:

Predicted attacks	0	2
Actual attacks		
0	2337	7374
2	212	2209

For R2L

In [62]:
```
Y_R2L_pred=clf_R2L.predict(X_R2L_test)
# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[62]:

Predicted attacks	0	3
Actual attacks		
0	9707	4
3	2573	312

For U2R:

In [63]:
```
Y_U2R_pred=clf_U2R.predict(X_U2R_test)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[63]:

Predicted attacks	0	4
Actual attacks		
0	9703	8
4	60	7

Cross Validation for each level: Accuracy, Precision, Recall, F-measure

DoS

In [64]:
```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
accuracy = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=1
```

```
0, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
precision = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=
10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precis
ion.std() * 2))
recall = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10,
scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std()
* 2))
f = cross_val_score(clf_DoS, X_DoS_test, Y_DoS_test, cv=10, scor
ing='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99639 (+/- 0.00341)
Precision: 0.99505 (+/- 0.00477)
Recall: 0.99665 (+/- 0.00483)
F-measure: 0.99585 (+/- 0.00392)
```

Probe

In [65]:

```
accuracy = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_tes
t, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
precision = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_tes
t, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precis
ion.std() * 2))
recall = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test,
cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std()
* 2))
f = cross_val_score(clf_Probe, X_Probe_test, Y_Probe_test, cv=1
0, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99571 (+/- 0.00328)
Precision: 0.99391 (+/- 0.00684)
Recall: 0.99267 (+/- 0.00405)
F-measure: 0.99329 (+/- 0.00512)
```

R2L

In [66]:

```
accuracy = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=1
0, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
precision = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=
10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precis
ion.std() * 2))
recall = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10,
scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std()
* 2))
f = cross_val_score(clf_R2L, X_R2L_test, Y_R2L_test, cv=10, scor
```

```
ing='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.97952 (+/- 0.00984)
Precision: 0.97216 (+/- 0.01610)
Recall: 0.96978 (+/- 0.01337)
F-measure: 0.97093 (+/- 0.01388)
```

U2R

In [67]:
```
accuracy = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_U2R, X_U2R_test, Y_U2R_test, cv=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

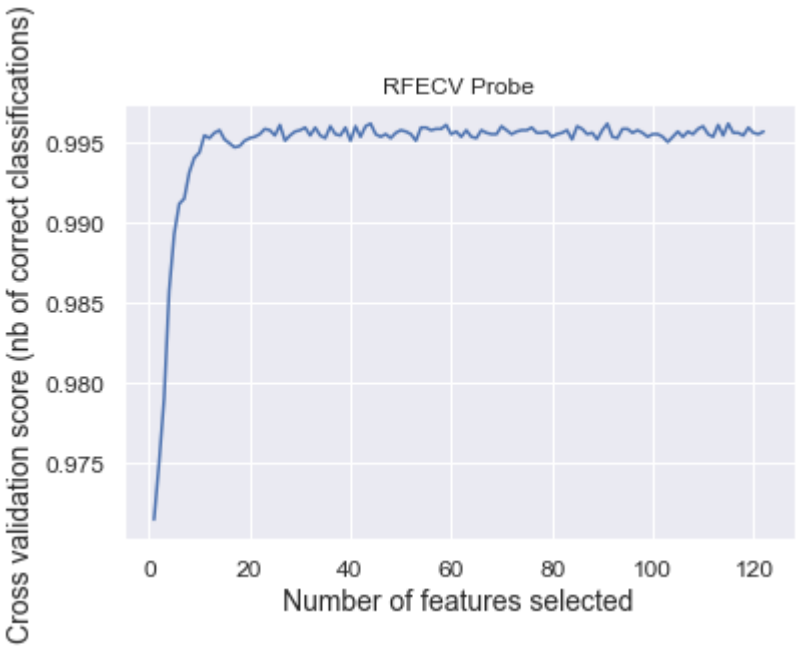
```
Accuracy: 0.99663 (+/- 0.00259)
Precision: 0.86481 (+/- 0.08952)
Recall: 0.91672 (+/- 0.10661)
F-measure: 0.88628 (+/- 0.07462)
```

RFECV for illustration

In [98]:
```
import matplotlib.pyplot as plt
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
```

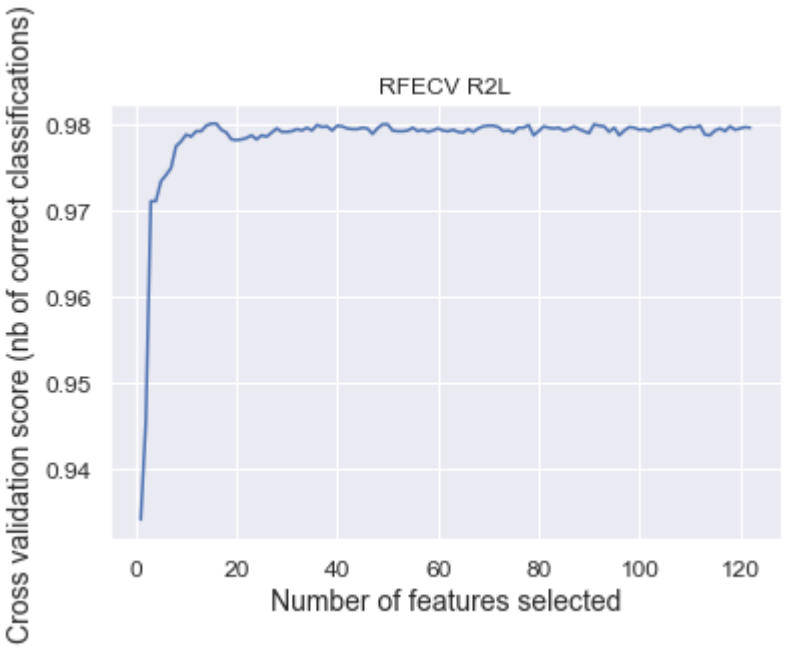
In [99]:
```
# Create the RFE object and compute a cross-validated score.
# The "accuracy" scoring is proportional to the number of correct
# classifications
rfecv_DoS = RFECV(estimator=clf_DoS, step=1, cv=10, scoring='accuracy')
rfecv_DoS.fit(X_DoS_test, Y_DoS_test)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.title('RFECV DoS')
plt.plot(range(1, len(rfecv_DoS.grid_scores_) + 1), rfecv_DoS.grid_scores_)
plt.show()
```

In [100]:
```
rfecv_Probe = RFECV(estimator=clf_Probe, step=1, cv=10, scoring
='accuracy')
rfecv_Probe.fit(X_Probe_test, Y_Probe_test)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classification
s)")
plt.title('RFECV Probe')
plt.plot(range(1, len(rfecv_Probe.grid_scores_) + 1), rfecv_Prob
e.grid_scores_)
plt.show()
```

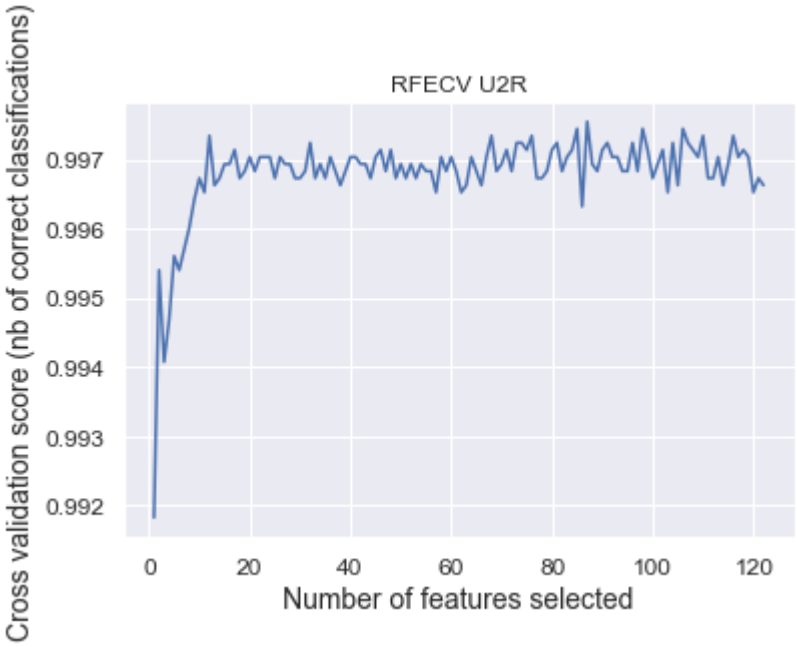


In [101]:
```
rfecv_R2L = RFECV(estimator=clf_R2L, step=1, cv=10, scoring='acc
uracy')
rfecv_R2L.fit(X_R2L_test, Y_R2L_test)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classification
s)")
plt.title('RFECV R2L')
plt.plot(range(1, len(rfecv_R2L.grid_scores_) + 1), rfecv_R2L.gr
```

```
id_scores_)
plt.show()
```



In [102]:
```
rfecv_U2R = RFECV(estimator=clf_U2R, step=1, cv=10, scoring='acc
uracy')
rfecv_U2R.fit(X_U2R_test, Y_U2R_test)
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classification
s)")
plt.title('RFECV U2R')
plt.plot(range(1, len(rfecv_U2R.grid_scores_) + 1), rfecv_U2R.gr
id_scores_)
plt.show()
```



Using 13 Features for each category

Confusion Matrices according to label

DoS

In [68]:
```
# reduce test dataset to 13 features, use only features describe
d in rfecolname_DoS etc.
X_DoS_test2=X_DoS_test[:,rfecolindex_DoS]
X_Probe_test2=X_Probe_test[:,rfecolindex_Probe]
X_R2L_test2=X_R2L_test[:,rfecolindex_R2L]
X_U2R_test2=X_U2R_test[:,rfecolindex_U2R]
X_U2R_test2.shape
```

Out[68]: (9778, 13)

In [69]:
```
Y_DoS_pred2=clf_rfeDoS.predict(X_DoS_test2)
# Create confusion matrix
pd.crosstab(Y_DoS_test, Y_DoS_pred2, rownames=['Actual attack
s'], colnames=['Predicted attacks'])
```

Out[69]:

Predicted attacks	0	1
Actual attacks		
0	9602	109
1	2625	4835

Probe

In [70]:
```
Y_Probe_pred2=clf_rfeProbe.predict(X_Probe_test2)
# Create confusion matrix
pd.crosstab(Y_Probe_test, Y_Probe_pred2, rownames=['Actual attac
ks'], colnames=['Predicted attacks'])
```

Out[70]:

Predicted attacks	0	2
Actual attacks		
0	8709	1002
2	944	1477

R2L

In [71]:
```
Y_R2L_pred2=clf_rfeR2L.predict(X_R2L_test2)
# Create confusion matrix
pd.crosstab(Y_R2L_test, Y_R2L_pred2, rownames=['Actual attack
s'], colnames=['Predicted attacks'])
```

Out[71]:

Predicted attacks	0	3
Actual attacks		
0	9649	62
3	2560	325

U2R

In [72]:
```
Y_U2R_pred2=clf_rfeU2R.predict(X_U2R_test2)
# Create confusion matrix
pd.crosstab(Y_U2R_test, Y_U2R_pred2, rownames=['Actual attacks'], colnames=['Predicted attacks'])
```

Out[72]:

Predicted attacks	0	4
Actual attacks		
0	9706	5
4	52	15

Cross Validation: Accuracy, Precision, Recall, F-measure sor above

For DOS

In [73]:
```
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='precision')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='recall')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std() * 2))
f = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test, cv=10, scoring='f1')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99732 (+/- 0.00251)
Precision: 0.99679 (+/- 0.00464)
Recall: 0.99705 (+/- 0.00356)
F-measure: 0.99692 (+/- 0.00288)
```

Probe

In [74]:
```
accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
precision = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precision.std() * 2))
recall = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_te
```

```
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std()
* 2))
f = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, c
v=10, scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99085 (+/- 0.00559)
Precision: 0.98674 (+/- 0.01180)
Recall: 0.98467 (+/- 0.01027)
F-measure: 0.98565 (+/- 0.00872)
```

R2L

In [76]:
```
accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test,
cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
precision = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test,
cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precis
ion.std() * 2))
recall = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv
=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std()
* 2))
f = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=10,
scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.97451 (+/- 0.00906)
Precision: 0.96683 (+/- 0.01316)
Recall: 0.96069 (+/- 0.01547)
F-measure: 0.96367 (+/- 0.01300)
```

U2R

In [77]:
```
accuracy = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test,
cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
precision = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test,
cv=10, scoring='precision_macro')
print("Precision: %0.5f (+/- %0.5f)" % (precision.mean(), precis
ion.std() * 2))
recall = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv
=10, scoring='recall_macro')
print("Recall: %0.5f (+/- %0.5f)" % (recall.mean(), recall.std()
* 2))
f = cross_val_score(clf_rfeU2R, X_U2R_test2, Y_U2R_test, cv=10,
scoring='f1_macro')
print("F-measure: %0.5f (+/- %0.5f)" % (f.mean(), f.std() * 2))
```

```
Accuracy: 0.99652 (+/- 0.00319)
Precision: 0.87747 (+/- 0.15709)
Recall: 0.89183 (+/- 0.17196)
F-measure: 0.87497 (+/- 0.11358)
```

Now Check with CV 2, 5, 10, 30, 50 fold

For DoS

In [78]:
```
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test,
cv=2, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
```

```
Accuracy: 0.99662 (+/- 0.00116)
```

In [79]:
```
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test,
cv=5, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
```

```
Accuracy: 0.99703 (+/- 0.00057)
```

In [80]:
```
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test,
cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
```

```
Accuracy: 0.99732 (+/- 0.00251)
```

In [81]:
```
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test,
cv=30, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
```

```
Accuracy: 0.99726 (+/- 0.00390)
```

In [82]:
```
accuracy = cross_val_score(clf_rfeDoS, X_DoS_test2, Y_DoS_test,
cv=50, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
```

```
Accuracy: 0.99703 (+/- 0.00599)
```

For Probe

In [83]:
```
accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_
test, cv=2, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
```

```
Accuracy: 0.99060 (+/- 0.00165)
```

In [84]:
```
accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_
test, cv=5, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accurac
y.std() * 2))
```

```
Accuracy: 0.99102 (+/- 0.00230)
```

In [85]:
```
accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=10, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
```

```
Accuracy: 0.99085 (+/- 0.00559)
```

In [86]:
```
accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=30, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
```

```
Accuracy: 0.99151 (+/- 0.00731)
```

In [87]:
```
accuracy = cross_val_score(clf_rfeProbe, X_Probe_test2, Y_Probe_test, cv=50, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
```

```
Accuracy: 0.99110 (+/- 0.01073)
```

For R2L

In [88]:
```
accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=2, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
```

```
Accuracy: 0.97134 (+/- 0.00174)
```

In [89]:
```
accuracy = cross_val_score(clf_rfeR2L, X_R2L_test2, Y_R2L_test, cv=5, scoring='accuracy')
print("Accuracy: %0.5f (+/- %0.5f)" % (accuracy.mean(), accuracy.std() * 2))
```

```
Accuracy: 0.97380 (+/- 0.00588)
```

Nov 9, 2019 (http://www.datasciencelovers.com/2019/11/)

Online Job Posting Analysis (http://www.datasciencelovers.com/machine-learning-projects/online-job-posting-analysis/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/online-job-posting-analysis/) Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/)** Tag **job posting analysis (http://www.datasciencelovers.com/tag/job-posting-analysis/), machine learning (http://www.datasciencelovers.com/tag/machine-learning/),**

NLP (http://www.datasciencelovers.com/tag/nlp/),
Text mining (http://www.datasciencelovers.com/tag/text-mining/)

Business Context:

The project seeks to understand the overall demand for labour in the Armenian online job market from the 19,000 job postings from 2004 to 2015 posted on Career Center, an Armenian human resource portal. Through text mining on this data, we will be able to understand the nature of the ever-changing job market, as well as the overall demand for labour in the Armenia economy. The data was originally scraped from a Yahoo! Mailing group.

Business Objectives:

Our main business objectives are to understand the dynamics of the labour market of Armenia using the online job portal post as a proxy. A secondary objective is to implement advanced text analytics as a proof of concept to create additional features such as enhanced search function that can add additional value to the users of the job portal.

So as a Data scientist you need to answer following business questions .

Job Nature and Company Profiles:

What are the types of jobs that are in demand in Armenia? How are the job natures changing over time?

Desired Characteristics and Skill-Sets:

What are the desired characteristics and skill -set of the candidates based on the job description dataset? How these are desired characteristics changing over time?

IT Job Classification:

Build a classifier that can tell us from the job description and company description whether a job is IT or not, so that this column can be automatically populated for new job postings. After doing so, understand what important factors are which drives this classification.

Similarity of Jobs:

Given a job title, find the 5 top jobs that are of a similar nature, based on the job post.

What should be our Text mining goal?

For the IT Job classification business question, you should aim to create supervised learning classification models that are able to classify based on the job text data accurately, is it an IT job.

On the business question of Job Nature and Company Profiles. Unsupervised learning techniques, such as topic modelling and other techniques such as term frequency counting will be applied to the data, including time period segmented dataset. Qualitative assessment will be done on the results to help us understand the job postings.

To understand the desired characteristics and skill -sets demanded by employers in the job ads, unsupervised learning methods such as K-means clustering will be used after appropriate dimension reduction.

For Job Queries business question, we propose exploring the usage of Latent Semantic Model and Matrix Similarity methods for information retrieval. The results will be assessed qualitatively. To return the top 5 most similar job posting, the job text data are vectorised using different models such as word2vec, and doc2vec and similarity scores are obtained using cosine similarity scores, ranked and returned as the answer which is then evaluated individually for relevance.

Data Understanding:

The data was obtained from Kaggle competition. Each row represents a job post. The dataset representation is tabular, but many of the columns are textual/unstructured in nature. Most notably, the columns job Description, Job Requirement, Required Qual, ApplicationP and AboutC are textual. The column job post is an amalgamation of these various textual columns.

Col Name	Description
Jobpost	The full text for the job post
Title	Title of the job
Announcement Code	Announcement code, which is some internal code and is usually missing.
Eligibility	Eligibility of the candidates.
StartDate	Start date of work.
Location	Employment location.
JobRequirement	Job requirements.
Salary	Job salary.
OpeningDate	Opening date of the job announcement.
Notes	Additional notes.
Attach	Attachments.
Month	The month of the announcement (derived from the field date).

Col Name	Description
Date	Date that the job was posted
Company	Company for the job
Term	Announcement code, which is some internal code and is usually missing.
Audience	Who can apply?
Duration	Duration of the employment.
JobDescription	Job Description.
RequiredQual	Required qualifications.
ApplicationP	Application procedure.
Deadline	Deadline for the job announcement.
AboutC	About the company.
Year	Year of the announcement (derived from the field date).
IT	TRUE if the job is an IT job.

Also provided sample job posting (attached with data set)

TITLE: Database Developer

LOCATION: Yerevan, Armenia

JOB DESCRIPTION: IUNetworks LLC is looking for a qualified Database Developer to design stable databases, according to the Company's needs. The incumbent will be responsible for developing, testing, improving and maintaining new and existing databases, ensuring the database systems run effectively and securely on a daily basis. He/ she will work closely with developers to ensure system consistency. He/ she will also collaborate with administrators and clients to provide technical support and identify new requirements.

JOB RESPONSIBILITIES:
- Design stable, reliable and effective databases;
- Optimize and maintain legacy systems;
- Modify databases according to requests and perform tests;
- Solve database usage issues and malfunctions;
- Liaise with developers to improve applications and establish best practices;
- Gather user requirements and identify new features;
- Develop technical and training manuals;
- Provide data management support to users;
- Ensure all database programs meet Company and performance requirements;
- Research and suggest new database products, services and protocols.

REQUIRED QUALIFICATIONS:
- BS in Computer Science or a relevant field;
- Proven work experience as a Database Developer;
- In-depth understanding of data management (e.g. permissions, recovery, security and monitoring);
- Knowledge of software development;
- Hands-on experience with SQL;

Let's develop a machine learning model for further analysis.

Online Job Posting Analysis

Import Library

In [1]:
```
import pandas as pd
import numpy as np
import nltk

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# spacy for lemmatization
import spacy
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

from nltk.corpus import stopwords
stop_words = stopwords.words('english')
```

Import Dataset

In [2]:
```
df = pd.read_csv("data job posts.csv")
df=df[df['Title'].isnull()==False]
df.reset_index(inplace=True)

df.head(5)
```

Out[2]:

	index	jobpost	date	Title	Company	AnnouncementCod
0	0	AMERIA Investment Consulting Company\nJOB TITL...	Jan 5, 2004	Chief Financial Officer	AMERIA Investment Consulting Company	Nal
1	1	International Research & Exchanges Board (IREX...	Jan 7, 2004	Full-time Community Connections Intern (paid i...	International Research & Exchanges Board (IREX)	Nal
2	2	Caucasus Environmental NGO Network (CENN)\nJOB...	Jan 7, 2004	Country Coordinator	Caucasus Environmental NGO Network (CENN)	Nal
3	3	Manoff Group\nJOB TITLE: BCC Specialist\nPOSI...	Jan 7, 2004	BCC Specialist	Manoff Group	Nal
4	4	Yerevan Brandy Company\nJOB TITLE: Software	Jan 10, 2004	Software Developer	Yerevan Brandy Company	Nal

D... 2004 Company

5 rows × 25 columns

In [3]:
```
# Get the basic info of DataFrame and perform programmatic assessment
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18973 entries, 0 to 18972
Data columns (total 25 columns):
index                18973 non-null int64
jobpost              18973 non-null object
date                 18973 non-null object
Title                18973 non-null object
Company              18973 non-null object
AnnouncementCode     1206 non-null object
Term                 7671 non-null object
Eligibility          4929 non-null object
Audience             640 non-null object
StartDate            9672 non-null object
Duration             10788 non-null object
Location             18948 non-null object
JobDescription       15090 non-null object
JobRequirment        16459 non-null object
RequiredQual         18496 non-null object
Salary               9614 non-null object
ApplicationP         18920 non-null object
OpeningDate          18275 non-null object
Deadline             18915 non-null object
Notes                2208 non-null object
AboutC               12460 non-null object
Attach               1548 non-null object
Year                 18973 non-null int64
Month                18973 non-null int64
IT                   18973 non-null bool
dtypes: bool(1), int64(3), object(21)
memory usage: 3.5+ MB
```

Find the job natute

In [4]:
```
#Finad the job nature and store in variable x
X=df['Title']
```

In [5]:
```
len(X)
```

Out[5]: 18973

Creating the corpus

In [7]:
```
import re
corpus = []
for i in range(0, len(X)):
    review = re.sub(r'\W', ' ', str(X[i]))
```

```
    review = re.sub(r'^br$', ' ', review)
    review = re.sub(r'\s+[a-z]\s+', ' ',review)
    review = re.sub(r'^[a-z]\s+', ' ', review)
    review = re.sub(r'\s+', ' ', review)
    corpus.append(review)
```

Remove all punctuations from title

In [8]:
```
# Remove punctuations from all title
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deac
c=True))  # deacc=True removes punctuations

data_words = list(sent_to_words(corpus))

data_words[0]
```

Out[8]:
```
['chief', 'financial', 'officer']
```

Define Stopwords, bigram model and lemmatization for title data

In [9]:
```
# Build the bigram and trigram models
bigram = gensim.models.Phrases(data_words, min_count=3, threshol
d=100) # higher threshold fewer phrases.
trigram = gensim.models.Phrases(bigram[data_words], threshold=10
0)

# Faster way to get a sentence clubbed as a trigram/bigram
bigram_mod = gensim.models.phrases.Phraser(bigram)
trigram_mod = gensim.models.phrases.Phraser(trigram)
```

In [10]:
```
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use','na','S
enior','new','branch','Junior','unit','department','Specialis
t','the','unit'])
```

In [11]:
```
# Define functions for stopwords, bigrams, trigrams and lemmatiz
ation
def remove_stopwords(texts):
    return [[word for word in simple_preprocess(str(doc)) if wor
d not in stop_words] for doc in texts]

def make_bigrams(texts):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB',
'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
```

```
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

In [12]:
```
# Remove Stop Words
data_words_nostops = remove_stopwords(data_words)

# Form Bigrams
data_words_bigrams = make_bigrams(data_words_nostops)

data_lemmatized = lemmatization(data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(data_lemmatized[:1])
```

```
[['chief', 'financial', 'officer']]
```

In [13]:
```
data_lemmatized[0:10]
```

Out[13]:
```
[['chief', 'financial', 'officer'],
 ['community', 'connection', 'intern', 'pay', 'internship'],
 ['country', 'coordinator'],
 ['specialist'],
 ['software', 'developer'],
 [],
 ['chief', 'accountant', 'finance', 'assistant'],
 ['pay', 'part', 'full_time', 'programmatic', 'intern'],
 ['assistant', 'manage', 'director'],
 ['program', 'assistant']]
```

In [14]:
```
# few null value is also coming so we are going to remove null from list
list2 = [x for x in data_lemmatized if x != []]
```

In [15]:
```
for i in range(len(list2)):
    list2[i] = ' '.join(list2[i])
```

In [16]:
```
list2[0:5]
```

Out[16]:
```
['chief financial officer',
 'community connection intern pay internship',
 'country coordinator',
 'specialist',
 'software developer']
```

Now we will select 1000 title for word cloud

In [17]:
```
titlecount = {}
for data in list2:
    words = nltk.word_tokenize(data)
    for word in words:
        if word not in titlecount.keys():
            titlecount[word] = 1
        else:
            titlecount[word] += 1
```

In [18]:
```
len(titlecount)
```

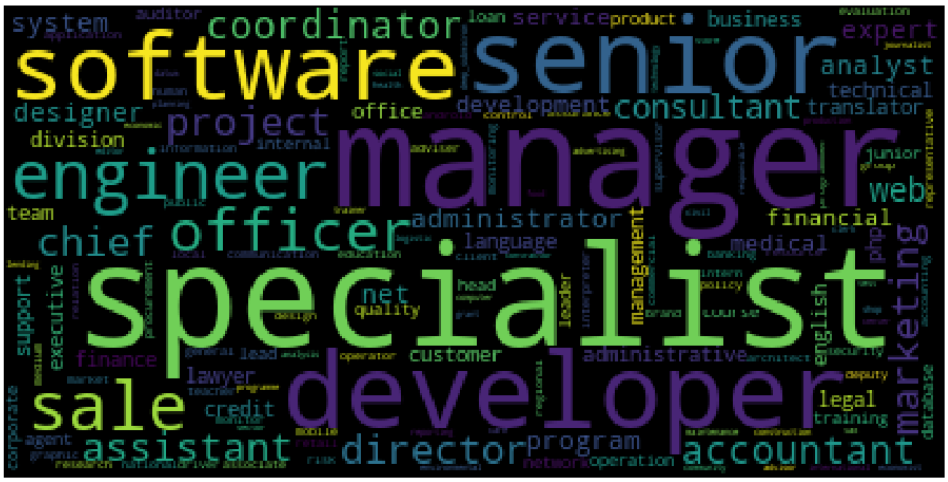
Out[18]: 1832

In [19]:
```
import heapq
# Selecting best 100 features
freq_words = heapq.nlargest(1000,titlecount,key=titlecount.get)
```

In [20]:
```
%%capture
#gather features
text = " ".join(freq_words)
```

In [21]:
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

In [22]:
```
wordcloud = WordCloud(stopwords=[],max_font_size=60).generate(te
xt)
plt.figure(figsize=(16,12))

# plot wordcloud in matplotlib
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



Job nature Change over the time

In [23]:
```
for i in range(len(data_lemmatized)):
    data_lemmatized[i] = ' '.join(data_lemmatized[i])
```

In [24]:
```
date_field=df['Year'].tolist()

len(date_field)
```

Out[24]: 18973

In [25]:
```
Job_year = pd.DataFrame(np.column_stack([data_lemmatized,date_fi
eld]),
                                columns=['Job_title','Year'])
```

In [26]:
```
Job_year.head(5)
```

Out[26]:

	Job_title	Year
0	chief financial officer	2004
1	community connection intern pay internship	2004
2	country coordinator	2004
3	specialist	2004
4	software developer	2004

In [27]:
```
Job_year.dtypes
```

Out[27]:
```
Job_title    object
Year         object
dtype: object
```

In [28]:
```
#Converting year to numeric value

Job_year['Year']=Job_year['Year'].astype('int')
```

In [29]:
```
Job_year.dtypes
```

Out[29]:
```
Job_title    object
Year          int32
dtype: object
```

In [30]:
```
Job_year.Year.value_counts()
```

Out[30]:
```
2012    2140
2015    2009
2013    2009
2014    1980
2008    1782
2011    1695
2007    1538
2010    1506
2009    1191
2005    1138
2006    1111
2004     874
Name: Year, dtype: int64
```

In [31]:
```
#We will devide year into 3 equidistant bins to find the job nature over the period

Job_year['Year_bins']=pd.cut(Job_year['Year'],3,labels=['Period1','Period2','Period3'])
```

In [32]:
```
Job_year.pivot_table(values='Year',index='Year_bins',aggfunc=['min','max','count'])
```

Out[32]:

	min	max	count
	Year	Year	Year
Year_bins			
Period1	2004	2007	4661
Period2	2008	2011	6174

Period3 2012 2015 8138

Job nature change over the period (2004 to 2007)

In [33]:
```
X1=Job_year[Job_year['Year_bins']=='Period1'].iloc[:,0]
```

In [34]:
```
# Selecting top 500 jobs for the wordcloud

titlecount = {}
for data in X1:
    words = nltk.word_tokenize(data)
    for word in words:
        if word not in titlecount.keys():
            titlecount[word] = 1
        else:
            titlecount[word] += 1
```

In [35]:
```
# import heapq
freq_words = heapq.nlargest(500,titlecount,key=titlecount.get)
```

In [36]:
```
%%capture
#gather features
text = " ".join(freq_words)
```

In [37]:
```
# Wordcloud need to draw for nature of job
wordcloud = WordCloud(stopwords=[],max_font_size=60).generate(te
xt)
plt.figure(figsize=(16,12))
# plot wordcloud in matplotlib
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

Job nature change over the period (2008 to 2011)

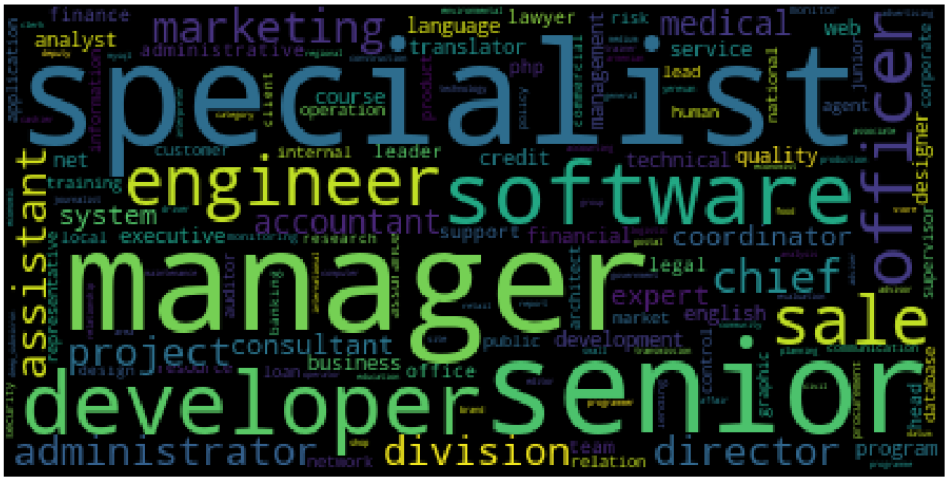
In [38]:
```
X2=Job_year[Job_year['Year_bins']=='Period2'].iloc[:,0]
```

In [39]:
```
# Selecting top 500 jobs for the wordcloud
```

In [39]:
```
# Selecting top 500 jobs for the wordcloud

titlecount = {}
for data in X2:
    words = nltk.word_tokenize(data)
    for word in words:
        if word not in titlecount.keys():
            titlecount[word] = 1
        else:
            titlecount[word] += 1

# import heapq
freq_words = heapq.nlargest(500,titlecount,key=titlecount.get)
```

In [40]:
```
%%capture
#gather features
text = " ".join(freq_words)
```

In [41]:
```
wordcloud = WordCloud(stopwords=[],max_font_size=60).generate(te
xt)
plt.figure(figsize=(16,12))
# plot wordcloud in matplotlib
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



Job nature change over the period (2012 to 2015)

In [42]:
```
X3=Job_year[Job_year['Year_bins']=='Period3'].iloc[:,0]
```

In [43]:
```
# Selecting top 500 jobs for the wordcloud

titlecount = {}
for data in X3:
    words = nltk.word_tokenize(data)
    for word in words:
        if word not in titlecount.keys():
            titlecount[word] = 1
        else:
            titlecount[word] += 1

# import heapq
freq_words = heapq.nlargest(500,titlecount,key=titlecount.get)
```

In [44]:
```
%%capture
#gather features
text = " ".join(freq_words)
```

In [45]:
```
wordcloud = WordCloud(stopwords=[],max_font_size=60).generate(te
xt)
plt.figure(figsize=(16,12))
# plot wordcloud in matplotlib
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

IT Job Classification

In [46]:
```
# Defining X
df['X1'] = df['Title'].str.cat(df['JobRequirment'], sep =" ").st
r.cat(df['RequiredQual'], sep =" ")
X=df['X1']

# Defining Y
di={False:0,True:1}
df['IT_y']=df['IT'].map(di)
y=df['IT_y']
```

In [47]:
```
# Creating the corpus
import re
corpus = []
for i in range(0, len(X)):
    review = re.sub(r'\W', ' ', str(X[i]))
    review = review.lower()
    review = re.sub(r'^br$', ' ', review)
    review = re.sub(r'\s+[a-z]\s+', ' ',review)
    review = re.sub(r'^[a-z]\s+', ' ', review)
    review = re.sub(r'\s+', ' ', review)
    corpus.append(review)
```

In [48]:
```
corpus[0]
```

Out[48]:
```
'chief financial officer supervises financial management and ad
ministrative staff including assigning responsibilities reviewi
```

ng employees work processes and products counseling employees g
iving performance evaluations and recommending disciplinary act
ion serves as member of management team participating in both s
trategic and operational planning for the company directs and o
versees the company financial management activities including e
stablishing and monitoring internal controls managing cash and
investments and managing the investment portfolio in collaborat
ion with the investment team leader this includes but is not li
mited to evaluation of investment risk concentration risk fund
deployment levels adequacy of loss and liquidity reserves assis
ts investment team in development of proper documentation and i
nternal systems directs and oversees the annual budgeting proce
ss including developing projections for financial planning and
preparing budgets prepares external and internal financial mana
gement reports such as audited financial statements tax returns
and reports for the board of directors and company staff develo
ps implements and maintains efficient and effective accounting
systems and controls to ensure compliance with national and int
ernational accounting standards and principles sufficiency of f
und accounting and comprehensiveness of data for reporting and
compliance requirements ensures contract compliance including i
nterpreting and monitoring contracts with clients submitting re
quired reports and monitoring covenants and other contract term
s oversees the design implementation and maintenance of compute
r based information system oversees records retention both manu
al and computer based and file maintenance activities serves as
company risk manager including evaluating loss exposure and obt
aining insurance as appropriate manages other administrative op
erations such as facilities management payroll administration o
ffice operations and administrative support monitors corporate
compliance with by laws and articles of incorporation regarding
corporate registration and reporting of fundraising operations
to perform this job successfully an individual must be able to
perform each essential duty satisfactorily the requirements lis
ted below are representative of the knowledge skill and or abil
ity required knowledge of generally accepted accounting princip
les local accounting standards and legislation state reporting
requirements pertaining to accounting principles and practices
of financial management and budgeting principles and practices
of financial systems design and analysis principles and practic
es of contract management records management and risk managemen
t principles and practices of management and supervision princi
ples and practices of information systems management ability to
apply sound fiscal and administrative practices to the company
activities plan organize and supervise the work of subordinate
employees including training them assigning and evaluating thei
r work and providing job performance feedback critically analyz
e fiscal and administrative policies practices procedures and s
ystems and recommend and implement changes as needed gather and
synthesize financial information from variety of sources and pr
esent it to variety of audiences with differing financial manag
ement and analysis expertise prepare detailed comprehensive fin
ancial reports including explanatory text operate ibm compatibl
e personal computer including word processing spreadsheet and d
atabase software applications operate specialized software appl
ications that support the financial management and budgeting fu
nctions qualifications minimum of 5 7 years accounting corporat
e finance banking experience including role as cfo excellent fi
nance and accounting technical skills coupled with demonstrated
knowledge of all key financial functions in an consulting compa
ny context accounting finance control treasury reserving and re
porting strong financial planning and analytical skills and exp

```
erience and the ability to work closely with and support the ce
o and other executives in strategic development and implementat
ion excellent leadership management and supervisory track recor
d of attracting selecting developing rewarding and retaining hi
gh caliber accounting and finance executive and teams who achie
ve business goals an undergraduate degree in finance business o
r other related discipline is required cpa cfa acca or other fi
nancial certification is highly preferred as is masters degree
in business administration accounting or finance fluency in eng
lish armenian and russian with outstanding writing skills excel
lent analytical communication teamwork interpersonal skills nee
d to be well organized and detail oriented as well as goal resu
lt driven and able to deal with complex issues '
```

In [49]:
```
from nltk.stem import PorterStemmer
#sentences = nltk.sent_tokenize(paragraph)
stemmer = PorterStemmer()

# Stemming
for i in range(len(corpus)):
    words = nltk.word_tokenize(corpus[i])
    words = [stemmer.stem(word) for word in words]
    corpus[i] = ' '.join(words)
```

In [50]:
```
corpus[0]
```

Out[50]:
```
'chief financi offic supervis financi manag and administr staff
includ assign respons review employe work process and product c
ounsel employe give perform evalu and recommend disciplinari ac
tion serv as member of manag team particip in both strateg and
oper plan for the compani direct and overse the compani financi
manag activ includ establish and monitor intern control manag c
ash and invest and manag the invest portfolio in collabor with
the invest team leader thi includ but is not limit to evalu of
invest risk concentr risk fund deploy level adequaci of loss an
d liquid reserv assist invest team in develop of proper documen
t and intern system direct and overse the annual budget process
includ develop project for financi plan and prepar budget prepa
r extern and intern financi manag report such as audit financi
statement tax return and report for the board of director and c
ompani staff develop implement and maintain effici and effect a
ccount system and control to ensur complianc with nation and in
tern account standard and principl suffici of fund account and
comprehens of data for report and complianc requir ensur contra
ct complianc includ interpret and monitor contract with client
submit requir report and monitor coven and other contract term
overse the design implement and mainten of comput base inform s
ystem overse record retent both manual and comput base and file
mainten activ serv as compani risk manag includ evalu loss expo
sur and obtain insur as appropri manag other administr oper suc
h as facil manag payrol administr offic oper and administr supp
ort monitor corpor complianc with by law and articl of incorpor
regard corpor registr and report of fundrais oper to perform th
i job success an individu must be abl to perform each essenti d
uti satisfactorili the requir list below are repres of the know
ledg skill and or abil requir knowledg of gener accept account
principl local account standard and legisl state report requir
pertain to account principl and practic of financi manag and bu
dget principl and practic of financi system design and analysi
principl and practic of contract manag record manag and risk ma
nag principl and practic of manag and supervis principl and pra
```

```
ctic of inform system manag abil to appli sound fiscal and admi
nistr practic to the compani activ plan organ and supervis the
work of subordin employe includ train them assign and evalu the
ir work and provid job perform feedback critic analyz fiscal an
d administr polici practic procedur and system and recommend an
d implement chang as need gather and synthes financi inform fro
m varieti of sourc and present it to varieti of audienc with di
ffer financi manag and analysi expertis prepar detail comprehen
s financi report includ explanatori text oper ibm compat person
comput includ word process spreadsheet and databas softwar appl
ic oper special softwar applic that support the financi manag a
nd budget function qualif minimum of 5 7 year account corpor fi
nanc bank experi includ role as cfo excel financ and account te
chnic skill coupl with demonstr knowledg of all key financi fun
ction in an consult compani context account financ control trea
suri reserv and report strong financi plan and analyt skill and
experi and the abil to work close with and support the ceo and
other execut in strateg develop and implement excel leadership
manag and supervisori track record of attract select develop re
ward and retain high calib account and financ execut and team w
ho achiev busi goal an undergradu degre in financ busi or other
relat disciplin is requir cpa cfa acca or other financi certif
is highli prefer as is master degre in busi administr account o
r financ fluenci in english armenian and russian with outstand
write skill excel analyt commun teamwork interperson skill need
to be well organ and detail orient as well as goal result drive
n and abl to deal with complex issu'
```

In [51]:
```
# Creating the Tf-Idf model directly
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features = 2000, min_df = 0.05,
max_df = 0.8, stop_words = stopwords.words('english'))

X = vectorizer.fit_transform(corpus).toarray()
```

In [52]:
```
X.shape
```

Out[52]:
```
(18973, 331)
```

Building Logistic Regression Model.

In [53]:
```
from sklearn.model_selection import train_test_split

train_x,test_x,train_y,test_y=train_test_split(X,
                                               y,
                                               test_size=.3,
                                               random_state=42)
```

In [54]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit( train_x, train_y )
```

Out[54]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_in
tercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear',
```

```
tol=0.0001,
          verbose=0, warm_start=False)
```

Evaluation of Model

In [55]:
```
#Predicting the test cases
from sklearn import metrics
test_accuracy=metrics.accuracy_score(test_y,logreg.predict(test_
x))
print('test_accuracy: ',test_accuracy)

train_accuracy=metrics.accuracy_score(train_y,logreg.predict(tra
in_x))
print('train_accuracy: ',train_accuracy)
```

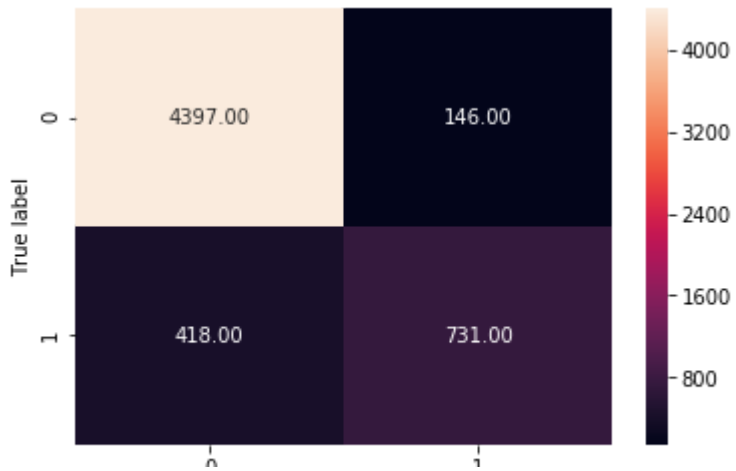
```
test_accuracy:  0.9009135628952917
train_accuracy:  0.91092538212484
```

In [56]:
```
print('AUC train :',metrics.roc_auc_score(train_y,logreg.predict
(train_x)))
print('AUC test :',metrics.roc_auc_score(test_y,logreg.predict(t
est_x)))
```

```
AUC train : 0.8128788197036333
AUC test : 0.8020340209126331
```

In [57]:
```
# Creating a confusion matrix

from sklearn import metrics

cm = metrics.confusion_matrix(test_y,
                              logreg.predict(test_x), [0,1] )
cm

import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline

sn.heatmap(cm, annot=True,  fmt='.2f', xticklabels = ["0", "1"]
, yticklabels = ["0", "1"] )
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[57]:
```
Text(0.5, 15.0, 'Predicted label')
```



In [58]:
```
from sklearn.metrics import classification_report
print(classification_report(test_y,logreg.predict(test_x)))
```

```
             precision    recall  f1-score   support

          0       0.91      0.97      0.94      4543
          1       0.83      0.64      0.72      1149

avg / total       0.90      0.90      0.90      5692
```

In [59]:
```
test_predicted_prob=pd.DataFrame(logreg.predict_proba(test_x))
[[1]]
test_predicted_prob.columns=['prob']
actual=test_y.reset_index()
actual.drop('index',axis=1,inplace=True)

# making a DataFrame with actual and prob columns
df_test_predict = pd.concat([actual, test_predicted_prob], axis=
1)
df_test_predict.columns = ['actual','prob']
df_test_predict.head()
```

Out[59]:

	actual	prob
0	0	0.026706
1	0	0.054805
2	1	0.982340
3	0	0.004979
4	0	0.052737

In [60]:
```
test_roc_like_df = pd.DataFrame()
test_temp = df_test_predict.copy()

for cut_off in np.linspace(0,1,50):
    test_temp['predicted'] = test_temp['prob'].apply(lambda x: 0
if x < cut_off else 1)
    test_temp['tp'] = test_temp.apply(lambda x: 1 if x['actual']
==1 and x['predicted']==1 else 0, axis=1)
    test_temp['fp'] = test_temp.apply(lambda x: 1 if x['actual']
==0 and x['predicted']==1 else 0, axis=1)
    test_temp['tn'] = test_temp.apply(lambda x: 1 if x['actual']
==0 and x['predicted']==0 else 0, axis=1)
    test_temp['fn'] = test_temp.apply(lambda x: 1 if x['actual']
==1 and x['predicted']==0 else 0, axis=1)
    sensitivity = test_temp['tp'].sum() / (test_temp['tp'].sum()
+ test_temp['fn'].sum())
    specificity = test_temp['tn'].sum() / (test_temp['tn'].sum()
+ test_temp['fp'].sum())

    accuracy=(test_temp['tp'].sum()+test_temp['tn'].sum()) / (te
st_temp['tp'].sum() + test_temp['fn'].sum()+test_temp['tn'].sum
() + test_temp['fp'].sum())

    test_roc_like_table = pd.DataFrame([cut_off, sensitivity, sp
```

```
ecificity,accuracy]).T
    test_roc_like_table.columns = ['cutoff', 'sensitivity', 'spe
cificity','accuracy']
    test_roc_like_df = pd.concat([test_roc_like_df, test_roc_lik
e_table], axis=0)
```

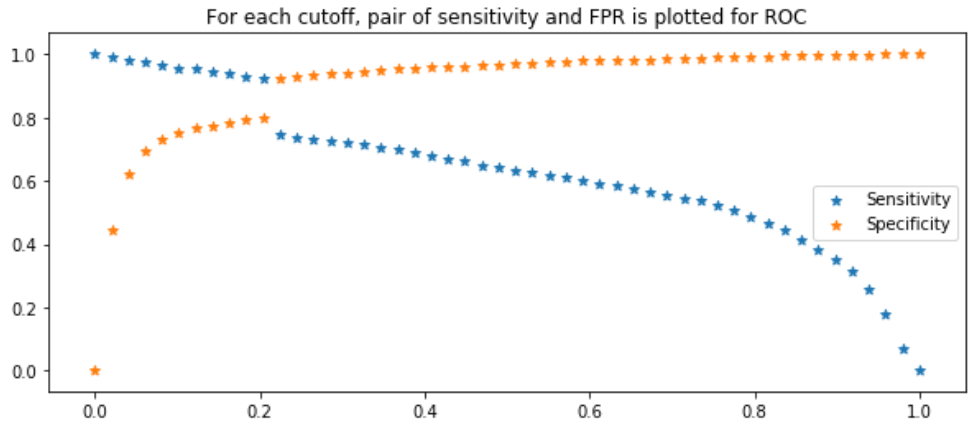
In [61]:
```
test_roc_like_df.head()
```

Out[61]:

	cutoff	sensitivity	specificity	accuracy
0	0.000000	1.000000	0.000000	0.201862
0	0.020408	0.989556	0.446621	0.556219
0	0.040816	0.979983	0.623157	0.695186
0	0.061224	0.973020	0.692274	0.748946
0	0.081633	0.965187	0.730795	0.778110

In [62]:
```
test_temp.sum()
plt.subplots(figsize=(10,4))
plt.scatter(test_roc_like_df['cutoff'], test_roc_like_df['sensit
ivity'], marker='*', label='Sensitivity')
plt.scatter(test_roc_like_df['cutoff'], test_roc_like_df['specif
icity'], marker='*', label='Specificity')
#plt.scatter(test_roc_like_df['cutoff'], 1-test_roc_like_df['spe
cificity'], marker='*', label='FPR')
plt.title('For each cutoff, pair of sensitivity and FPR is plott
ed for ROC')
plt.legend()
```

Out[62]:
```
<matplotlib.legend.Legend at 0x57747b8>
```



In [63]:
```
#Finding ideal cut-off for checking if this remains same in OOS
validation
test_roc_like_df['total'] = test_roc_like_df['sensitivity'] + te
st_roc_like_df['accuracy']
test_roc_like_df[test_roc_like_df['total']==test_roc_like_df['to
tal'].max()]
```

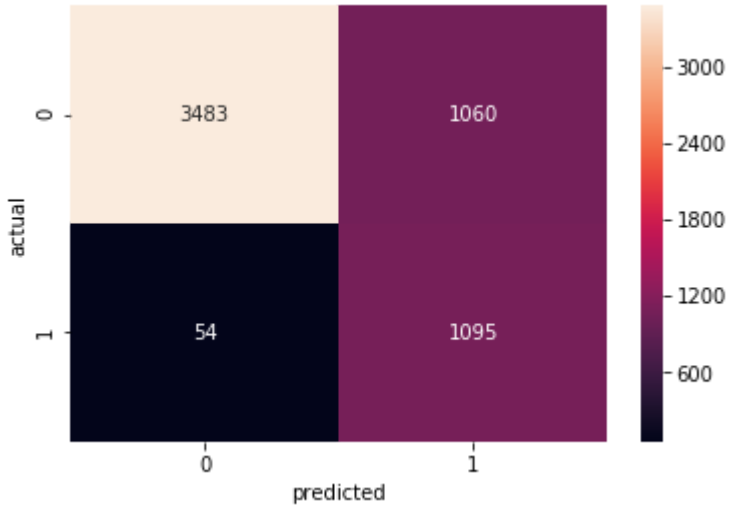
Out[63]:

	cutoff	sensitivity	specificity	accuracy	total
0	0.122449	0.953003	0.766674	0.804287	1.757289

In [64]:
```
df_test_predict['predicted'] = df_test_predict['prob'].apply(lam
bda x: 1 if x > 0.122449 else 0)
```

```
import seaborn as sns
sns.heatmap(pd.crosstab(df_test_predict['actual'], df_test_predi
ct['predicted']), annot=True, fmt='.0f')
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x57c62e8>



In [65]:
```
accuracy=metrics.accuracy_score(df_test_predict.actual, df_test_
predict.predicted)
print('Accuracy: ',round(accuracy,2))
```

```
Accuracy:  0.8
```

In [66]:
```
from sklearn.metrics import classification_report
print(classification_report(df_test_predict.actual, df_test_pred
ict.predicted))
```

```
             precision    recall  f1-score   support

          0       0.98      0.77      0.86      4543
          1       0.51      0.95      0.66      1149

avg / total       0.89      0.80      0.82      5692
```

Building Naive Bayes Model

In [67]:
```
from sklearn.naive_bayes import GaussianNB

nb_clf=GaussianNB()

nb_clf.fit(train_x,train_y)
```

Out[67]: GaussianNB(priors=None)

Model Evelutation

In [68]:
```
#Predicting the test cases
from sklearn import metrics
test_accuracy=metrics.accuracy_score(test_y,nb_clf.predict(test_
x))
```

```
print('test_accuracy: ',test_accuracy)

train_accuracy=metrics.accuracy_score(train_y,nb_clf.predict(tra
in_x))
print('train_accuracy: ',train_accuracy)
```

```
test_accuracy:  0.808854532677442
train_accuracy:  0.8018221519463896
```

In [69]:
```
print('AUC train :',metrics.roc_auc_score(train_y,nb_clf.predict
(train_x)))
print('AUC test :',metrics.roc_auc_score(test_y,nb_clf.predict(t
est_x)))
```

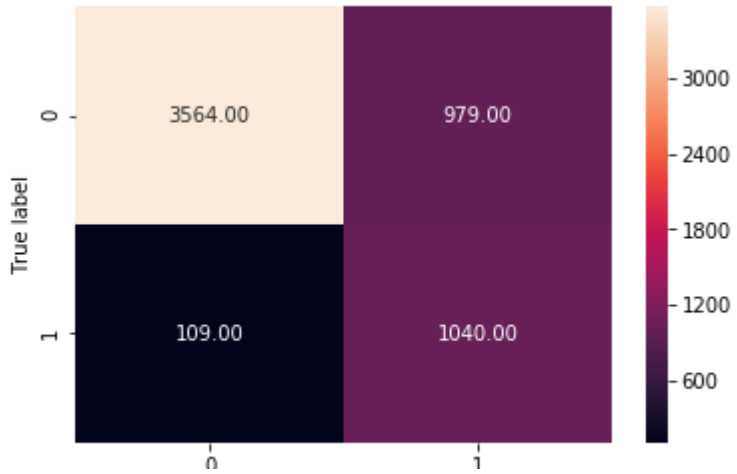
```
AUC train : 0.8365890688804225
AUC test : 0.8448192659371134
```

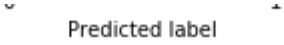
In [70]:
```
from sklearn.metrics import classification_report
print(classification_report(test_y,nb_clf.predict(test_x)))
```

```
             precision    recall  f1-score   support

          0       0.97      0.78      0.87      4543
          1       0.52      0.91      0.66      1149

avg / total       0.88      0.81      0.82      5692
```

In [71]:
```
# Creating a confusion matrix

from sklearn import metrics

cm = metrics.confusion_matrix(test_y,
                              nb_clf.predict(test_x), [0,1] )
cm

import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline

sn.heatmap(cm, annot=True,  fmt='.2f', xticklabels = ["0", "1"]
, yticklabels = ["0", "1"] )
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[71]: Text(0.5, 15.0, 'Predicted label')





Apply Random forrest model

In [72]:
```
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import GridSearchCV
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\wei
ght_boosting.py:29: DeprecationWarning: numpy.core.umath_tests
is an internal NumPy module and should not be imported. It will
be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [73]:
```
param_grid={'n_estimators':[100, 200, 400, 600, 800]}

tree=GridSearchCV(RandomForestClassifier(oob_score=False,warm_st
art=True),param_grid,cv=5,n_jobs=-1)
tree.fit(train_x,train_y)
```

Out[73]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_w
eight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes
=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jo
bs=1,
            oob_score=False, random_state=None, verbose=0, warm
_start=True),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'n_estimators': [100, 200, 400, 600, 800]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score
='warn',
       scoring=None, verbose=0)
```

In [75]:
```
tree.best_params_
```

Out[75]:
```
{'n_estimators': 800}
```

In [76]:
```
radm_clf=RandomForestClassifier(oob_score=True,n_estimators=800,
n_jobs=-1)
radm_clf.fit(train_x,train_y)
```

Out[76]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, crite
rion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes
=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=800, n_j
obs=-1,
            oob_score=True, random_state=None, verbose=0, warm_
start=False)
```

Model Eveluation

In [77]:
```
#Predicting the test cases
from sklearn import metrics
test_accuracy=metrics.accuracy_score(test_y,radm_clf.predict(tes
t_x))
print('test_accuracy: ',test_accuracy)

train_accuracy=metrics.accuracy_score(train_y,radm_clf.predict(t
rain_x))
print('train_accuracy: ',train_accuracy)
```

```
test_accuracy:  0.9214687280393534
train_accuracy:  0.9704841502898878
```

In [78]:
```
print('AUC train :',metrics.roc_auc_score(train_y,radm_clf.predi
ct(train_x)))
print('AUC test :',metrics.roc_auc_score(test_y,radm_clf.predict
(test_x)))
```

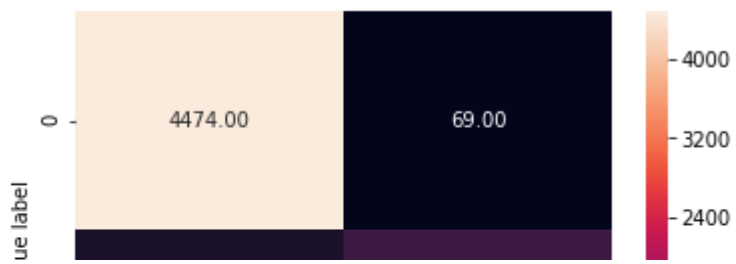
```
AUC train : 0.924904214559387
AUC test : 0.8279150375667612
```

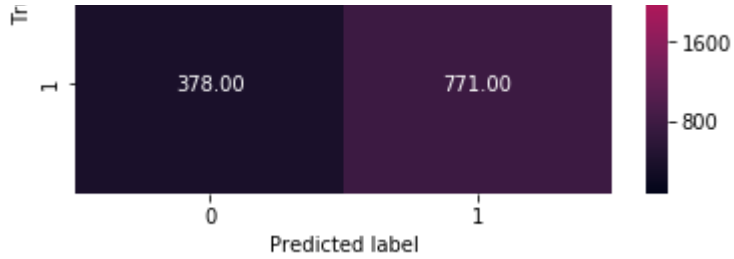
In [79]:
```
from sklearn.metrics import classification_report
print(classification_report(test_y,radm_clf.predict(test_x)))
```

```
             precision    recall  f1-score   support

          0       0.92      0.98      0.95      4543
          1       0.92      0.67      0.78      1149

avg / total       0.92      0.92      0.92      5692
```

In [80]:
```
# Creating a confusion matrix

from sklearn import metrics

cm = metrics.confusion_matrix(test_y,
                              radm_clf.predict(test_x), [0,1] )
cm

import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline

sn.heatmap(cm, annot=True,  fmt='.2f', xticklabels = ["0", "1"]
, yticklabels = ["0", "1"] )
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[80]: Text(0.5, 15.0, 'Predicted label')





In [81]:
```
test_predicted_prob=pd.DataFrame(radm_clf.predict_proba(test_x))
[[1]]
test_predicted_prob.columns=['prob']
actual=test_y.reset_index()
actual.drop('index',axis=1,inplace=True)

# making a DataFrame with actual and prob columns
df_test_predict = pd.concat([actual, test_predicted_prob], axis=
1)
df_test_predict.columns = ['actual','prob']
df_test_predict.head()
```

Out[81]:

	actual	prob
0	0	0.01125
1	0	0.05625
2	1	1.00000
3	0	0.00375
4	0	0.04500

In [82]:
```
test_roc_like_df = pd.DataFrame()
test_temp = df_test_predict.copy()

for cut_off in np.linspace(0,1,50):
    test_temp['predicted'] = test_temp['prob'].apply(lambda x: 0
if x < cut_off else 1)
    test_temp['tp'] = test_temp.apply(lambda x: 1 if x['actual']
==1 and x['predicted']==1 else 0, axis=1)
    test_temp['fp'] = test_temp.apply(lambda x: 1 if x['actual']
==0 and x['predicted']==1 else 0, axis=1)
    test_temp['tn'] = test_temp.apply(lambda x: 1 if x['actual']
==0 and x['predicted']==0 else 0, axis=1)
    test_temp['fn'] = test_temp.apply(lambda x: 1 if x['actual']
==1 and x['predicted']==0 else 0, axis=1)
    sensitivity = test_temp['tp'].sum() / (test_temp['tp'].sum()
+ test_temp['fn'].sum())
    specificity = test_temp['tn'].sum() / (test_temp['tn'].sum()
+ test_temp['fp'].sum())

    accuracy=(test_temp['tp'].sum()+test_temp['tn'].sum()) / (te
st_temp['tp'].sum() + test_temp['fn'].sum()+test_temp['tn'].sum
() + test_temp['fp'].sum())

    test_roc_like_table = pd.DataFrame([cut_off, sensitivity, sp
ecificity,accuracy]).T
    test_roc_like_table.columns = ['cutoff', 'sensitivity', 'spe
cificity','accuracy']
    test_roc_like_df = pd.concat([test_roc_like_df, test_roc_lik
e_table], axis=0)
```

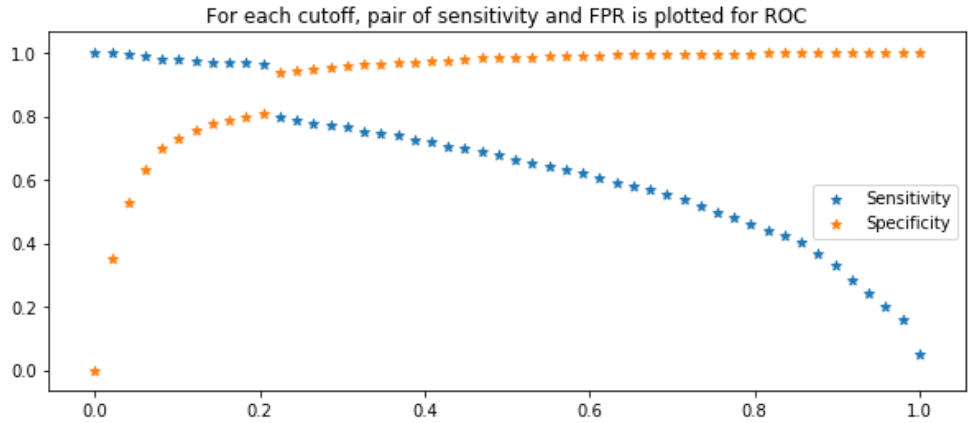
In [83]:
```
test_roc_like_df.head(5)
```

Out[83]:

	cutoff	sensitivity	specificity	accuracy
0	0.000000	1.000000	0.000000	0.201862
0	0.020408	0.998259	0.351530	0.482080
0	0.040816	0.994778	0.530046	0.623858
0	0.061224	0.988686	0.633502	0.705200
0	0.081633	0.981723	0.696676	0.754216

In [84]:
```
test_temp.sum()
plt.subplots(figsize=(10,4))
plt.scatter(test_roc_like_df['cutoff'], test_roc_like_df['sensitivity'], marker='*', label='Sensitivity')
plt.scatter(test_roc_like_df['cutoff'], test_roc_like_df['specificity'], marker='*', label='Specificity')
#plt.scatter(test_roc_like_df['cutoff'], 1-test_roc_like_df['specificity'], marker='*', label='FPR')
plt.title('For each cutoff, pair of sensitivity and FPR is plotted for ROC')
plt.legend()
```

Out[84]: <matplotlib.legend.Legend at 0x1d754c88>



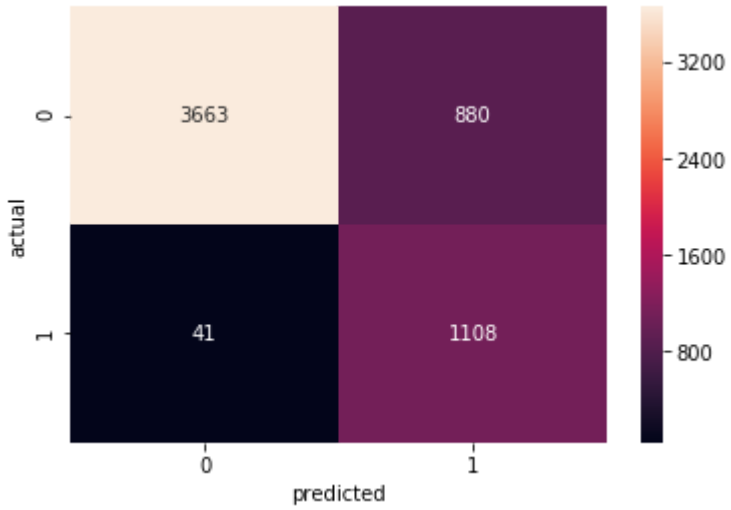
In [85]:
```
## Finding ideal cut-off for checking if this remains same in OOS validation
test_roc_like_df['total'] = test_roc_like_df['sensitivity'] + test_roc_like_df['specificity']
test_roc_like_df[test_roc_like_df['total']==test_roc_like_df['total'].max()]
```

Out[85]:

	cutoff	sensitivity	specificity	accuracy	total
0	0.204082	0.964317	0.806295	0.838194	1.770612

In [86]:
```
df_test_predict['predicted'] = df_test_predict['prob'].apply(lambda x: 1 if x > 0.204082 else 0)

import seaborn as sns
sns.heatmap(pd.crosstab(df_test_predict['actual'], df_test_predict['predicted']), annot=True, fmt='.0f')
```

Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x1d77c5c0>



In [87]:
```
accuracy=metrics.accuracy_score(df_test_predict.actual, df_test_
predict.predicted)
print('Accuracy: ',round(accuracy,2))
```

```
Accuracy:  0.84
```

In [88]:
```
from sklearn.metrics import classification_report
print(classification_report(df_test_predict.actual, df_test_pred
ict.predicted))
```

```
             precision    recall  f1-score   support

          0       0.99      0.81      0.89      4543
          1       0.56      0.96      0.71      1149

avg / total       0.90      0.84      0.85      5692
```

Text Clustring

In [89]:
```
train_x.shape
```

Out[89]:
```
(13281, 331)
```

In [90]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
```

In [91]:
```
title=list2
title[0:5]
```

Out[91]:
```
['chief financial officer',
 'community connection intern pay internship',
 'country coordinator',
 'specialist',
 'software developer']
```

In [92]:
```
# Creating the Tf-Idf model directly
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features = 2000, min_df = 0.01,
```

```
max_df = 0.9, stop_words = stopwords.words('english'))
X = vectorizer.fit_transform(title).toarray()
```

In [93]:
```
X.shape
```

Out[93]:
```
(16537, 48)
```

Cluster Error

In [94]:
```
cluster_range = range( 1, 21 )
cluster_errors = []

for num_clusters in cluster_range:
    clusters = KMeans( num_clusters )
    clusters.fit(X)
    cluster_errors.append( clusters.inertia_ )
```

In [95]:
```
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "clu
ster_errors": cluster_errors } )

clusters_df[0:21]
```

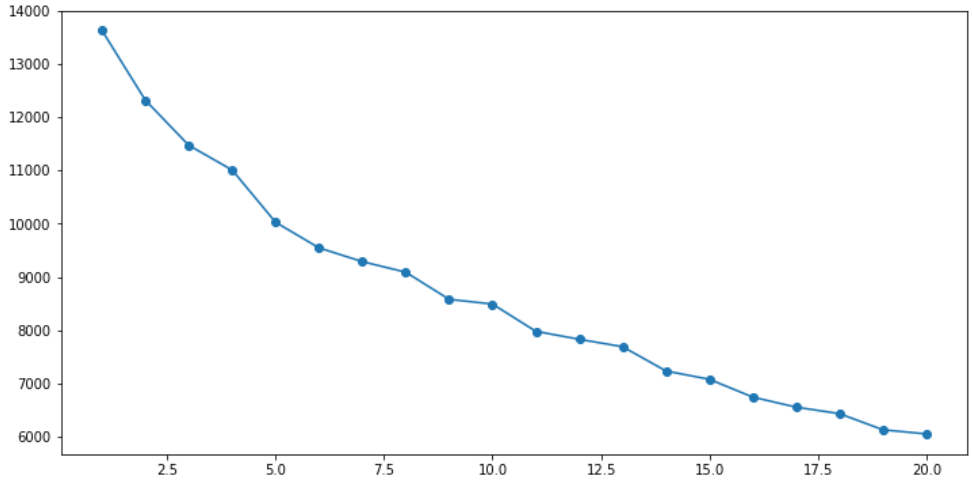
Out[95]:

	num_clusters	cluster_errors
0	1	13631.991020
1	2	12320.649888
2	3	11474.188001
3	4	11009.692194
4	5	10027.822592
5	6	9546.478743
6	7	9287.663126
7	8	9088.867504
8	9	8578.701953
9	10	8489.231648
10	11	7976.736505
11	12	7826.614321
12	13	7686.765777
13	14	7231.006232
14	15	7074.110078
15	16	6739.365930
16	17	6550.441574
17	18	6427.778388
18	19	6124.412782
19	20	6047.016351

In [96]:
```
# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors,
marker = "o" )
```

Out[96]: [<matplotlib.lines.Line2D at 0x1f3e1940>]



Define Silhouette Coefficient

In [97]:
```
from sklearn import metrics
```

In [ ]:
```
# calculate SC for K=3 through K=12
k_range = range(2, 21)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=1)
    km.fit(X)
    scores.append(metrics.silhouette_score(X, km.labels_))
```

Nov 9, 2019 (http://www.datasciencelovers.com/2019/11/)

Bank Review and Complaints Analysis (http://www.datasciencelovers.com/machine-learning-projects/bank-review-and-complaints-analysis/)

By **Datasciencelovers (http://www.datasciencelovers.com/author/ashwini/)** in **(http://www.datasciencelovers.com/machine-learning-projects/bank-review-and-complaints-analysis/) Machine learning projects (http://www.datasciencelovers.com/category/machine-learning-projects/)** Tag **bank complaint analysis (http://www.datasciencelovers.com/tag/bank-complaint-analysis/), Bank review analysis (http://www.datasciencelovers.com/tag/bank-review-analysis/), NLP (http://www.datasciencelovers.com/tag/nlp/), Text mining (http://www.datasciencelovers.com/tag/text-mining/)**

Business Problem

Central banks collecting information about customer satisfaction with the services provided by different bank. Also collects the information about the complaints.

- Bank users give ratings and write reviews about services on central bank websites. These reviews and ratings help to banks evaluate services provided and take necessary to action improve customer service. While ratings are useful to convey the overall experience, they do not convey the context which led a reviewer to that experience.

- If we look at only the rating, it is difficult to guess why the user rated the service as 4 stars. However, after reading the review, it is not difficult to identify that the review talks about good “service” and “expectations”.

So the Business Requirement is to analyze customer reviews and predict customer satisfaction with the reviews. It should include following tasks.

- Data processing
- Key positive words/negative words (most frequent words)
- Classification of reviews into positive, negative and neutral
- Identify key themes of problems (using clustering, topic models)
- Predicting star ratings using reviews
- Perform intent analysis

Datasets:

BankReviews.xlsx.

The data is a detailed dump of customer reviews/complaints (~500) of different services at different banks.

Data Dictionary:

- Date (Day the review was posted)
- Stars (1–5 rating for the business)
- Text (Review text),
- Bank name

Let’s develop a machine learning model for further analysis.

Import Important library.

In [3]:
```
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
```

In [5]:
```
df = pd.read_excel("BankReviews.xlsx")
df.head()
```

Out[5]:

	Date	Stars	Reviews	BankName
0	2017-04-10	5	Great job, Wyndham Capital! Each person was pr...	Wyndham Capital Mortgage
1	2017-02-10	5	Matthew Richardson is professional and helpful...	Wyndham Capital Mortgage
2	2017-08-21	5	We had a past experience with Wyndham Mortgage...	Wyndham Capital Mortgage
3	2017-12-17	5	We have been dealing with Brad Thomka from the...	Wyndham Capital Mortgage
4	2016-05-27	5	I can't express how grateful I am for the supp...	Wyndham Capital Mortgage

In [6]:
```
# Dropping the irrelevant variables
df.drop(['Date','BankName'],axis=1,inplace=True)
```

In [7]:
```
df.head(3)
```

Out[7]:

	Stars	Reviews
0	5	Great job, Wyndham Capital! Each person was pr...
1	5	Matthew Richardson is professional and helpful...
2	5	We had a past experience with Wyndham Mortgage...

In [8]:
```
df.shape
```

Out[8]:
```
(505, 2)
```

Define X and Y

In [9]:
```
X,y=df.Reviews,df.Stars
```

In [10]:
```
X.head()
```

Out[10]:
```
0    Great job, Wyndham Capital! Each person was pr...
1    Matthew Richardson is professional and helpful...
2    We had a past experience with Wyndham Mortgage...
3    We have been dealing with Brad Thomka from the...
4    I can't express how grateful I am for the supp...
```

```
Name: Reviews, dtype: object
```

In [11]:
```
y.head()
```

Out[11]:
```
0    5
1    5
2    5
3    5
4    5
Name: Stars, dtype: int64
```

In [12]:
```
# Pickling the dataset

import pickle
with open('Review.pickle','wb') as f:
    pickle.dump(X,f)

with open('sent.pickle','wb') as f:
    pickle.dump(y,f)
```

In [13]:
```
# Unpickling dataset
X_in = open('Review.pickle','rb')
y_in = open('sent.pickle','rb')
X = pickle.load(X_in)
y = pickle.load(y_in)
```

In [14]:
```
# Creating the corpus
import re
corpus = []
for i in range(0, len(X)):
    review = re.sub(r'\W', ' ', str(X[i]))
    review = review.lower()
    review = re.sub(r'^br$', ' ', review)
    review = re.sub(r'\s+[a-z]\s+', ' ',review)
    review = re.sub(r'^[a-z]\s+', '', review)
    review = re.sub(r'\s+', ' ', review)
    corpus.append(review)
```

In [15]:
```
X[0]
```

Out[15]:
```
'Great job, Wyndham Capital! Each person was professional and helpe
d us move through our refinance process smoothly. Thank you!'
```

In [16]:
```
corpus[1]
```

Out[16]:
```
'matthew richardson is professional and helpful he helped us find t
he correct product for our mortgage thank you very much for the exc
ellent service matthew '
```

In [17]:
```
from nltk.stem import WordNetLemmatizer
# sentences = nltk.sent_tokenize(paragraph)
lemmatizer = WordNetLemmatizer()

# Lemmatization
for i in range(len(corpus)):
    words = nltk.word_tokenize(corpus[i])
    words = [lemmatizer.lemmatize(word) for word in words]
    corpus[i] = ' '.join(words)
```

Creating Tf - Idf model

In [18]:
```
# Creating the Tf-Idf model
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features = 2000, min_df = 0.05, ma
x_df = 0.8, stop_words = stopwords.words('english'))
X = vectorizer.fit_transform(corpus).toarray()
```

In [19]:
```
X
```

Out[19]:
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.3324
5169,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.12340384, 0.        , ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.1137738 , 0.0843
0181,
        0.1272345 ]])
```

In [20]:
```
X.shape
```

Out[20]:
```
(505, 137)
```

Apply Logistic regression

In [22]:
```
from sklearn.model_selection import train_test_split

train_x,test_x,train_y,test_y=train_test_split(X,
                                               y,
                                               test_size=.3,
                                               random_state=42)
```

In [23]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit( train_x, train_y )
```

Out[23]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interc
ept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_j
obs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=
0.0001,
          verbose=0, warm_start=False)
```

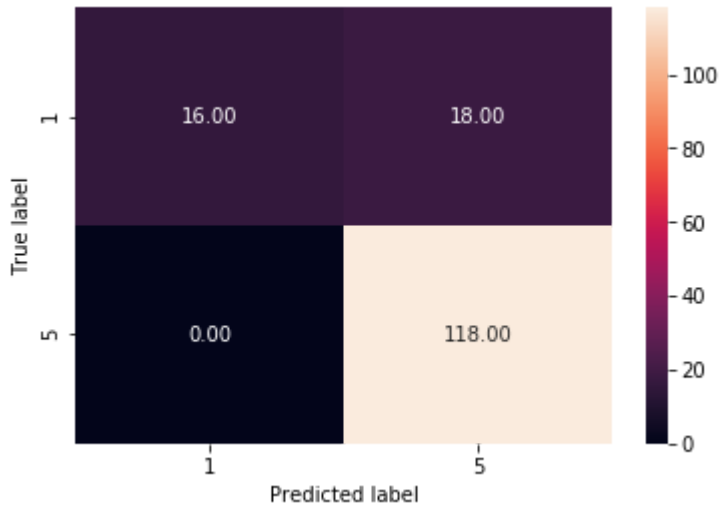
Model Accuracy Eveluation

In [25]:
```
#Evelution of train and test accuracy
from sklearn import metrics
test_accuracy=metrics.accuracy_score(test_y,logreg.predict(test_x))
print('test_accuracy: ',test_accuracy)

train_accuracy=metrics.accuracy_score(train_y,logreg.predict(train_
x))
print('train_accuracy: ',train_accuracy)
```

```
test_accuracy:  0.881578947368421
train_accuracy:  0.9178470254957507
```

In [28]:
```
# Creating a confusion matrix

from sklearn import metrics

cm = metrics.confusion_matrix(test_y,
                              logreg.predict(test_x), [1,5] )
cm

import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline

sn.heatmap(cm, annot=True,  fmt='.2f', xticklabels = ["1", "5"] , y
ticklabels = ["1", "5"] )
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

Out[28]: Text(0.5, 15.0, 'Predicted label')



As we can see from confusion matrix our model is not over fitted

Final Prediction

In [29]:
```
#Concatenating final prediction with original data set(all obersvat
ions):
```

```
tons):

pred_stars=pd.DataFrame(logreg.predict(X),columns=['predicted_star
s'])

testfile = pd.concat([df, pred_stars], axis=1)

testfile.head(10)
```

Out[29]:

	Stars	Reviews	predicted_stars
0	5	Great job, Wyndham Capital! Each person was pr...	5
1	5	Matthew Richardson is professional and helpful...	5
2	5	We had a past experience with Wyndham Mortgage...	5
3	5	We have been dealing with Brad Thomka from the...	5
4	5	I can't express how grateful I am for the supp...	5
5	5	I had the pleasure of working with Wyndham Cap...	5
6	5	My experience with Mattison was beyond greatly...	5
7	5	Patrick answered all my questions by email imm...	5
8	5	I loved working with this group of people! The...	5
9	5	Great web interface for both the loan applicat...	5

Final Submission in excel file

In [30]:
```
#Exporting testfile to csv for final submission

testfile.to_csv('Review_submission.csv',index=False)
```

In [ ]:
```

```

Happy Learning....