Introduction

This is the first contact with the featuring engineering process and its impact in a ML pipeline. Feature engineering is one of the most important step of the process of developing prediction models.
The experimental dataset used is the HR Analytics Dataset. It includes explanatory variables of around 15k employees of a large company. The goal of the case study is to model the probability of attrition (employees leaving, either on their own or because they got fired) of each employee, as well as to understand which variables are the most important ones and need to be addressed right away. The results obtained will be helpful for the management in order to understand what changes they should make to their workplace to get most of their employees to stay.

In [78]:

```
from dataset import Dataset as dataset
from sklearn.linear_model import LogisticRegression
from typing import List
from skrebate import ReliefF
from pandas import read_csv
from sklearn import linear_model
from sklearn import metrics
from sklearn.model_selection import cross_val_score, cross_val_predict, cross_validate
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [ ]:

```
pip install nbconvert
```

Data Loading

In [80]:

```
data = pd.read_csv("turnover.csv")
```

In [81]:

```
data.head()
```

Out[81]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last
0	0.38	0.53	2	157	3	0	1	
1	0.80	0.86	5	262	6	0	1	
2	0.11	0.88	7	272	4	0	1	
3	0.72	0.87	5	223	5	0	1	
4	0.37	0.52	2	159	3	0	1	

Removing Null Values

Creating one more dataframe that conatins our original dataset

In [82]:

```
dataa = pd.DataFrame(data)
```

In [83]:

```
data1 = data.dropna(axis = 0)
```

In [84]:

```
print("Length of old dataframe:", len(data), "\nLength of new dataframe:",
       len(data1), "\nNumber of rows with NA values: ",
       (len(data)-len(data1)))
```

```
Length of old dataframe: 14999
Length of new dataframe: 14999
Number of rows with NA values:  0
```

No null values present

Renaming few columns for better understanding of our dataset

In [85]:

```
data = data.rename(columns={'satisfaction_level': 'Satisfaction',
                        'promotion_last_5years': 'Promotion',
                        'sales' : 'Department',
                            'average_montly_hours' : 'average_monthly_hours'
                        })
data.head()
```

Out[85]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	left	Promotion	Depart
0	0.38	0.53	2	157	3	0	1	0	
1	0.80	0.86	5	262	6	0	1	0	
2	0.11	0.88	7	272	4	0	1	0	
3	0.72	0.87	5	223	5	0	1	0	
4	0.37	0.52	2	159	3	0	1	0	

Checking the data types of all features

In [86]:

```
data.dtypes
```

Out[86]:

```
Satisfaction             float64
last_evaluation          float64
number_project             int64
average_monthly_hours      int64
time_spend_company         int64
Work_accident              int64
left                       int64
Promotion                  int64
Department                object
salary                    object
dtype: object
```

Checking Skewness of every column

Will remove skewness if present because there are some methods that assume that the residuals are normal

In [87]:

```
data.skew(axis = 0)
```

Out[87]:

```
Satisfaction            -0.476360
last_evaluation         -0.026622
number_project           0.337706
average_monthly_hours    0.052842
time_spend_company       1.853319
Work_accident            2.021149
left                     1.230043
Promotion                6.636968
dtype: float64
```

Skewness of first four columns seems fine, have to change skewness of column time_spend_company , other columns are categorical in nature so there will be some skewness always

In [90]:

```
skew_log = np.log(data.time_spend_company)
skew_log.describe()
```

Out[90]:

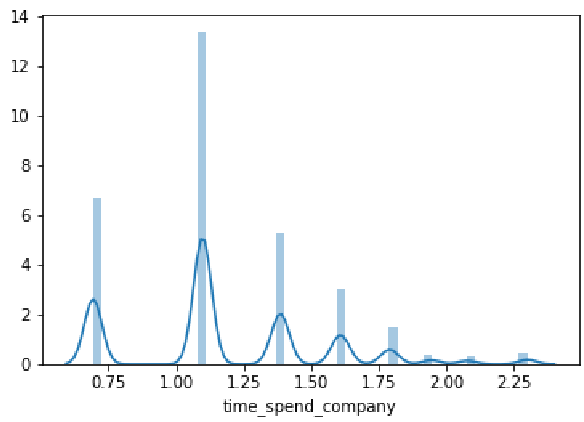
```
count    14999.000000
mean         1.181700
std          0.362584
min          0.693147
25%          1.098612
50%          1.098612
75%          1.386294
max          2.302585
Name: time_spend_company, dtype: float64
```

In [91]:

```
sns.distplot(skew_log)
```

Out[91]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2130cdb65c8>
```



In [92]:

```
skew_log.skew()
```

Out[92]:

```
0.5885330284719315
```

Skewness of column time_spend_company is reduced to 0.5885330284719315

We are going to map the salary column to substitute each categorical value in this column into numbers because later we have to scale these values and also Logistic Regression supports only numerical values

In [93]:

```
data["salary"] = data.salary.map({'low':1, 'medium':2,'high':3})
```

Scaling

Scaling all numerical columns to normalise the data within 0 to 1 range to ensure that the algorithm treats them with equal importance and it also speeds up the calculations in an algorithm.

In [94]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

data[['Satisfaction', 'last_evaluation', 'number_project', 'average_monthly_hours',
      'time_spend_company' , 'Work_accident', 'left', 'Promotion', 'salary'
     ]] = scaler.fit_transform(data[['Satisfaction', 'last_evaluation', 'number_project', 'average_
monthly_hours',
                                            'time_spend_company' , 'Work_accident', 'left',
'Promotion', 'salary' ]])

data
```

Out[94]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	left	Promotion	De
0	0.318681	0.265625	0.0	0.285047	0.125	0.0	1.0	0.0	
1	0.780220	0.781250	0.6	0.775701	0.500	0.0	1.0	0.0	
2	0.021978	0.812500	1.0	0.822430	0.250	0.0	1.0	0.0	
3	0.692308	0.796875	0.6	0.593458	0.375	0.0	1.0	0.0	
4	0.307692	0.250000	0.0	0.294393	0.125	0.0	1.0	0.0	
...	...	...	...	...	...	...	...	...	
14994	0.340659	0.328125	0.0	0.257009	0.125	0.0	1.0	0.0	
14995	0.307692	0.187500	0.0	0.299065	0.125	0.0	1.0	0.0	
14996	0.307692	0.265625	0.0	0.219626	0.125	0.0	1.0	0.0	
14997	0.021978	0.937500	0.8	0.859813	0.250	0.0	1.0	0.0	
14998	0.307692	0.250000	0.0	0.289720	0.125	0.0	1.0	0.0	

14999 rows × 10 columns

In [95]:

```
data.dtypes
```

Out[95]:

```
Satisfaction             float64
last_evaluation          float64
number_project           float64
average_monthly_hours    float64
time_spend_company       float64
Work_accident            float64
left                     float64
Promotion                float64
Department                object
salary                   float64
dtype: object
```

Changing column Department to category as i have to do one hot encoding of categorical features

In [96]:

```
data['Department']=data.Department.astype('category')
```

One Hot Encoding

Doing One hot encoding because most machine learning algorithms require numerical input and output variables

In [97]:

```
one_hot_encoder = OneHotEncoder()

#Including all categorical features in cat_col
cat_col = data.columns[data.dtypes=='category'].tolist()

#Including all other features apart from categorical in num_col
num_col = data.columns[data.dtypes!='category'].tolist()
num_col.remove('left')

dummy_data = pd.concat([data[num_col].reset_index(drop=True),
                  pd.DataFrame(one_hot_encoder.fit_transform(data[cat_col]).toarray(), columns=one_
hot_encoder.get_feature_names(cat_col)).reset_index(drop=True),
                  data['left'].reset_index(drop=True)], axis=1)
dummy_data.head()
```

Out[97]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	salary	Dep
0	0.318681	0.265625	0.0	0.285047	0.125	0.0	0.0	0.0	
1	0.780220	0.781250	0.6	0.775701	0.500	0.0	0.0	0.5	
2	0.021978	0.812500	1.0	0.822430	0.250	0.0	0.0	0.5	
3	0.692308	0.796875	0.6	0.593458	0.375	0.0	0.0	0.0	
4	0.307692	0.250000	0.0	0.294393	0.125	0.0	0.0	0.0	

Splitting dataset into test and train

In [98]:

```
from sklearn.model_selection import train_test_split

target_name = 'left'
X = dummy_data.drop('left', axis=1)
y=dummy_data[target_name]

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.15, random_state=1, stratify=y)

X_train.head()
```

Out[98]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	salary
13475	0.538462	0.859375	0.2	0.271028	1.000	0.0	0.0	1.0
9105	0.813187	0.656250	0.6	0.719626	0.000	1.0	0.0	0.0
11226	0.527473	1.000000	0.4	0.612150	1.000	0.0	0.0	0.0
4042	0.681319	0.828125	0.2	0.691589	0.250	0.0	0.0	0.5
4781	0.692308	0.531250	0.2	0.247664	0.125	0.0	0.0	0.5

Checking Baseline

In [99]:

```
model = LogisticRegression(random_state = 99)
model.fit(X_train,y_train)
predictions = model.predict(X_test)
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.82      0.94      0.87      1714
         1.0       0.62      0.34      0.44       536

    accuracy                           0.79      2250
   macro avg       0.72      0.64      0.66      2250
weighted avg       0.77      0.79      0.77      2250
```

Feature Engineering

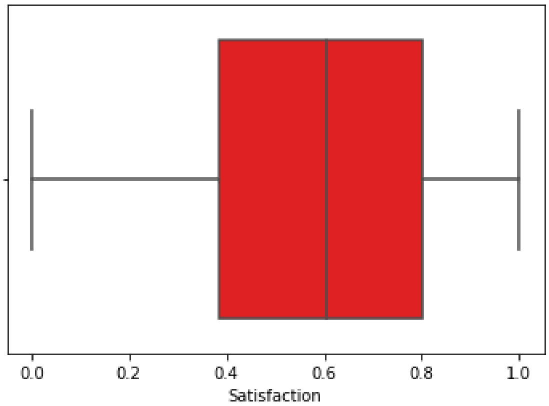
Outliers Detection

In [100]:

```
import seaborn as sns
sns.boxplot(x=data['Satisfaction'], color = 'r')
```

Out[100]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2130cdad8c8>
```



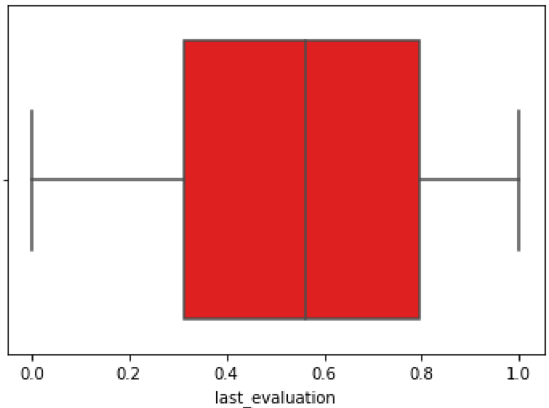
No outlier is present in column Satisfaction

In [101]:

```
sns.boxplot(x=data['last_evaluation'],color='r')
#No outlier is present in column last_evaluation
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2130cb41c88>
```



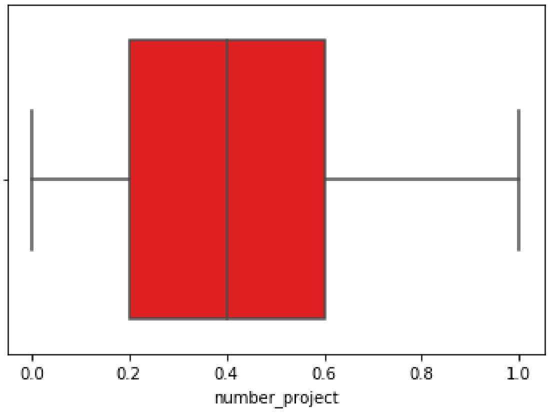
In [102]:

```
sns.boxplot(x=data['number_project'],color ='r')
```

```
sns.boxplot(x=data['number_project'],color = 'r')
#No outlier is present in column number_project
```

Out[102]:

<matplotlib.axes._subplots.AxesSubplot at 0x2130c9d9188>

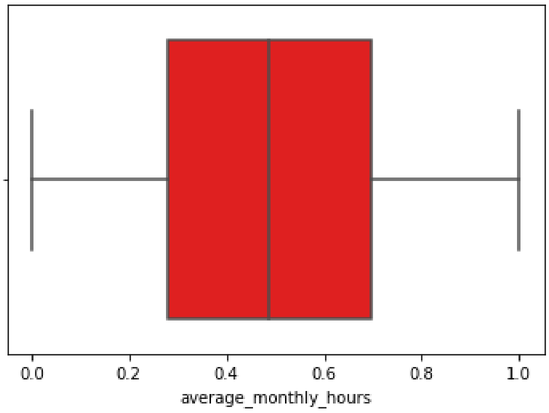


In [103]:

```
sns.boxplot(x=data['average_monthly_hours'],color = 'r')
#No outlier is present in column average_monthly_hours
```

Out[103]:

<matplotlib.axes._subplots.AxesSubplot at 0x2130ca8e888>

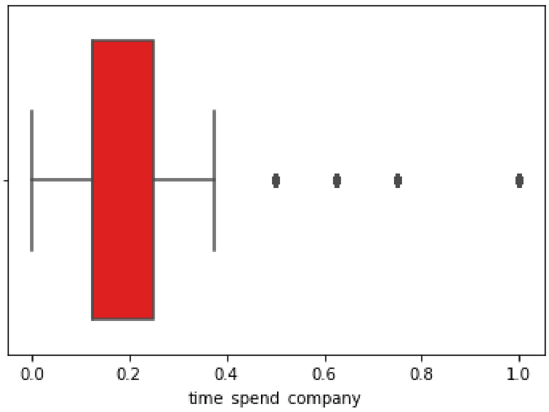


In [104]:

```
sns.boxplot(x=data['time_spend_company'], color = 'r')
#There are some outliers in time_spend_company
```

Out[104]:

<matplotlib.axes._subplots.AxesSubplot at 0x2130a8ae288>



So how many people spent more than around 6 years in company

In [105]:

```
out = data[data['time_spend_company'] > 0.6]
out.count()
```

Out[105]:

```
Satisfaction             564
last_evaluation          564
number_project           564
average_monthly_hours    564
time_spend_company       564
Work_accident            564
left                     564
Promotion                564
Department               564
salary                   564
dtype: int64
```

564 people spent more than 6 years in company

Are they leaving after spending so many years in company?

In [106]:

```
left_group = out.groupby('left')

left_group.count()
```

Out[106]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	Departme
left								
0.0	564	564	564	564	564	564	564	56

Nobody left after spending so many years in one company, probably removing these people from our dataset will create a balance

In [107]:

```
outliers=[]
def remove_outlier(data_in, time_spend_company):
    Q1 = data_in[time_spend_company].quantile(0.25)
    Q3 = data_in[time_spend_company].quantile(0.75)
    IQR = Q3-Q1
    lower_bound  = Q1-1.5*IQR
    upper_bound = Q3+1.5*IQR
    data_out = data_in.loc[(data_in[time_spend_company] > lower_bound) &
(data_in[time_spend_company] < upper_bound)]

    print("{} outliers removed".format(len(data_in) - len(data_out)))
    return data_out
```

In [108]:

```
data = remove_outlier(data,'time_spend_company')
```

```
1282 outliers removed
```

Converting Department column from string values to numerical values to check accuracy and also because Logistic Regression supports numerical values

In [109]:

```
data.Department.unique()
```

Out[109]:

```
[sales, accounting, hr, technical, support, management, IT, product_mng, marketing, RandD]
Categories (10, object): [sales, accounting, hr, technical, ..., IT, product_mng, marketing, RandD
]
```

In [110]:

```
data["Department"] = data.Department.map({'sales':1, 'accounting':2,'hr':3, 'technical':4, 'support
':5 , 'management':6,
                                        'IT':7, 'product_mng':8, 'marketing':9, 'RandD':0})
```

After removing outliers , again have to split new dataset in train and test, #splitting the original data that is not one hot encoded

In [111]:

```
target_name = 'left'
X = data.drop('left', axis=1)


y=data[target_name]

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.15, random_state=1, stratify=y)

X_train.head()
```

Out[111]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	Departm
746	0.296703	0.218750	0.0	0.168224	0.125	0.0	0.0	
8691	0.439560	0.328125	0.2	0.285047	0.125	0.0	0.0	
9865	0.956044	0.234375	0.6	0.509346	0.000	1.0	0.0	
2253	0.142857	0.250000	0.8	0.369159	0.250	0.0	0.0	
6944	0.857143	0.187500	0.4	0.172897	0.000	0.0	0.0	

Cross Validating to prevent overfitting

In [112]:

```
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split,KFold,RandomizedSearchCV,GridSearchCV
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
```

In [113]:

```
kfold = KFold(n_splits=5, random_state=99)
model = LogisticRegression(C=5)
results = cross_val_score(model ,X_train, y_train, cv=kfold)
results
```

Out[113]:

```
array([0.82418525, 0.83576329, 0.83662093, 0.81389365, 0.84212784])
```

Checking model accuracy and F1 score

In [114]:

```
model = LogisticRegression(random_state = 99)
model.fit(X_train,y_train)
predictions = model.predict(X_test)
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.87      0.91      0.89      1554
         1.0       0.68      0.57      0.62       504

    accuracy                           0.83      2058
   macro avg       0.77      0.74      0.75      2058
weighted avg       0.82      0.83      0.82      2058
```

Will do some Binning For instance, the 'time_spend_company' can be represented by different numbers (1, 2, 3, ...). We will then create another "bucketized" feature for some numerical columns because some features are much more fine grained than we need Represented 'time_spend_company' in numbers because Logistic Regression only supports numerical values

In [115]:

```
#Checking values present in time_spend_company

data['time_spend_company'].unique()
```

Out[115]:

```
array([0.125, 0.25 , 0.375, 0.   ])
```

In [116]:

```
#Binning
criteria = [data['number_project'].between(0,0.2), data['number_project'].between(0.2,0.4), data['n
umber_project'].between(0.4,1)]
values = ["1","2","3"]

data['number_project_cat'] = np.select(criteria, values, 0)

#data['number_project_cat'] =
data['number_project'].map({0:"Low",0.2:"Low",0.4:"Medium",0.6:"Medium",0.8:"High",1:"Very_High"}).
pe('category')
data['time_spend_company_cat'] = data['time_spend_company'].map({0.:"1",0.125:"2",0.25:"3",0.375:"4
"}).astype('category')
```

In [117]:

```
data.head()
```

Out[117]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	left	Promotion	Depart
0	0.318681	0.265625	0.0	0.285047	0.125	0.0	1.0	0.0	
2	0.021978	0.812500	1.0	0.822430	0.250	0.0	1.0	0.0	
3	0.692308	0.796875	0.6	0.593458	0.375	0.0	1.0	0.0	
4	0.307692	0.250000	0.0	0.294393	0.125	0.0	1.0	0.0	
5	0.351648	0.218750	0.0	0.266355	0.125	0.0	1.0	0.0	

In [118]:

```
data.dtypes
```

Out[118]:

```
Satisfaction             float64
last_evaluation          float64
number_project           float64
average_monthly_hours    float64
time_spend_company       float64
Work_accident            float64
```

```
left                        float64
Promotion                   float64
Department                 category
salary                      float64
number_project_cat           object
time_spend_company_cat     category
dtype: object
```

Converting column created after binning 'number_project_cat' into category because we have to do one hot encoding of a categorical column 'number_project_cat'

In [119]:

```
#Converting column created after binning 'number_project_cat' into category
data['number_project_cat']=data.number_project_cat.astype('category')
```

In [120]:

```
data.head()
```

Out[120]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	left	Promotion	Depart
0	0.318681	0.265625	0.0	0.285047	0.125	0.0	1.0	0.0	
2	0.021978	0.812500	1.0	0.822430	0.250	0.0	1.0	0.0	
3	0.692308	0.796875	0.6	0.593458	0.375	0.0	1.0	0.0	
4	0.307692	0.250000	0.0	0.294393	0.125	0.0	1.0	0.0	
5	0.351648	0.218750	0.0	0.266355	0.125	0.0	1.0	0.0	

One Hot Encoding

Doing One hot encoding because most machine learning algorithms require numerical input and output variables to do a better job in prediction

In [121]:

```
one_hot_encoder = OneHotEncoder()

#Including all categorical features in cat_col
cat_col = data.columns[data.dtypes=='category'].tolist()

#Including all other features apart from categorical in num_col
num_col = data.columns[data.dtypes!='category'].tolist()
num_col.remove('left')

dummy1_data = pd.concat([data[num_col].reset_index(drop=True),
                  pd.DataFrame(one_hot_encoder.fit_transform(data[cat_col]).toarray(), columns=one_
hot_encoder.get_feature_names(cat_col)).reset_index(drop=True),
                  data['left'].reset_index(drop=True)], axis=1)
dummy1_data.head()
```

Out[121]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	salary	Dep
0	0.318681	0.265625	0.0	0.285047	0.125	0.0	0.0	0.0	
1	0.021978	0.812500	1.0	0.822430	0.250	0.0	0.0	0.5	
2	0.692308	0.796875	0.6	0.593458	0.375	0.0	0.0	0.0	
3	0.307692	0.250000	0.0	0.294393	0.125	0.0	0.0	0.0	
4	0.351648	0.218750	0.0	0.266355	0.125	0.0	0.0	0.0	

5 rows × 26 columns

Spliting Test and train

In [122]:

```
target_name = 'left'
X2 = dummy1_data.drop('left', axis=1)


y2=dummy1_data[target_name]

X2_train, X2_test, y2_train, y2_test = train_test_split(X2,y2,test_size=0.15, random_state=42, stra
tify=y)

X2_train.head()
```

Out[122]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	salary
1611	0.000000	0.640625	0.8	0.869159	0.375	0.0	0.0	0.5
8823	0.307692	0.718750	0.2	0.481308	0.375	0.0	0.0	0.5
2383	0.670330	0.421875	0.2	0.275701	0.250	1.0	0.0	0.0
11306	0.714286	1.000000	0.4	0.714953	0.375	0.0	0.0	0.0
10612	0.395604	0.296875	0.4	0.514019	0.000	0.0	0.0	0.0

5 rows × 25 columns

Cross Validating to prevent over-fitting

In [123]:

```
kfold = KFold(n_splits=5, random_state=99)
model = LogisticRegression(C=5)
results = cross_val_score(model ,X2_train, y2_train, cv=kfold)
results
```

Out[123]:

```
array([0.90651801, 0.90909091, 0.8983705 , 0.8957976 , 0.90733591])
```

In [124]:

```
model = LogisticRegression(random_state = 99)
model.fit(X2_train,y2_train)
predictions = model.predict(X2_test)
print(classification_report(y2_test,predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.92      0.93      1554
         1.0       0.78      0.83      0.81       504

    accuracy                           0.90      2058
   macro avg       0.86      0.88      0.87      2058
weighted avg       0.90      0.90      0.90      2058
```

Seems a improvement in results , lets see some feature importance

Filtering

Important features of the LogisticRegression

In [125]:

```
lm = linear_model.LogisticRegression(max_iter=10000, penalty = 'none')
lm.fit(X2_train, y2_train)
```

Out[125]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='auto', n_jobs=None, penalty='none',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Ploting the feature importance of the LogisticRegression models, this code is taken from https://stackoverflow.com/a/47191103)

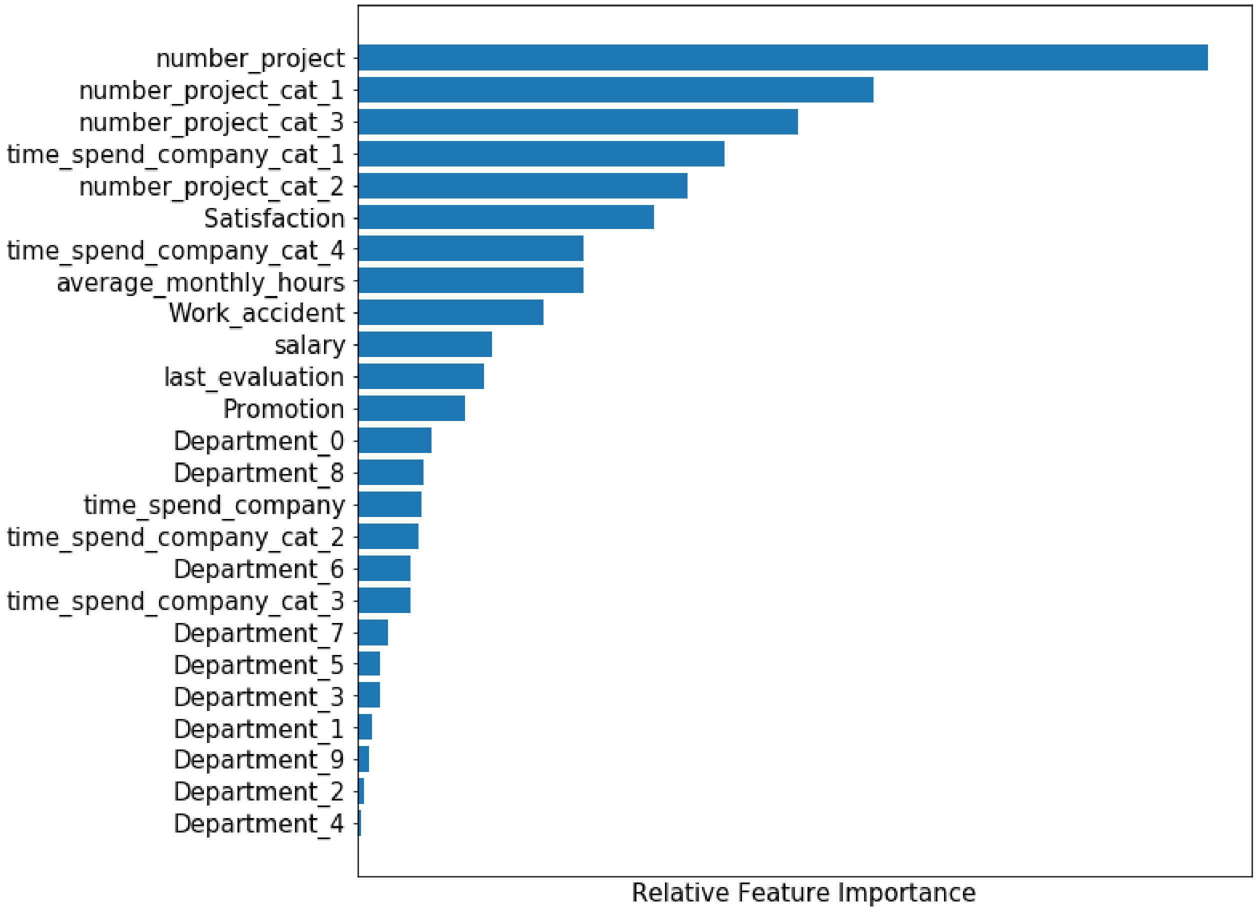
In [126]:

```
def get_feature_importance(clf):
    feature_importance = abs(clf.coef_[0])
    feature_importance = 100.0 * (feature_importance / feature_importance.max())
    sorted_idx = np.argsort(feature_importance)
    pos = np.arange(sorted_idx.shape[0]) + .5

    featfig = plt.figure(figsize=(10,10))
    featax = featfig.add_subplot(1, 1, 1)
    featax.barh(pos, feature_importance[sorted_idx], align='center')
    featax.set_yticks(pos)
    featax.set_xticks([])
    featax.set_yticklabels(np.array(X2.columns)[sorted_idx], fontsize=15)
    featax.set_xlabel('Relative Feature Importance', fontsize=15)
```

In [128]:

```
get_feature_importance(lm)
```



In [129]:

```
data.head()
```

Out[129]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	left	Promotion	Depart
0	0.318681	0.265625	0.0	0.285047	0.125	0.0	1.0	0.0	
2	0.021978	0.812500	1.0	0.822430	0.250	0.0	1.0	0.0	
3	0.692308	0.796875	0.6	0.593458	0.375	0.0	1.0	0.0	
4	0.307692	0.250000	0.0	0.294393	0.125	0.0	1.0	0.0	
5	0.351648	0.218750	0.0	0.266355	0.125	0.0	1.0	0.0	

Droping features that are not very important, Keeping top 12 features

In [130]:

```
drop_department = dummy1_data.drop(axis =0,columns =[ 'Department_4', 'Department_2',
       'Department_9', 'Department_1', 'Department_3', 'time_spend_company_cat_3',
'time_spend_company_cat_2',
       'Department_5', 'Department_7', 'Department_6',
       'Department_0', 'Department_8', 'time_spend_company'])
drop_department.head()
```

Out[130]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	Work_accident	Promotion	salary	number_project_cat_1	nur
0	0.318681	0.265625	0.0	0.285047	0.0	0.0	0.0	1.0	
1	0.021978	0.812500	1.0	0.822430	0.0	0.0	0.5	0.0	
2	0.692308	0.796875	0.6	0.593458	0.0	0.0	0.0	0.0	
3	0.307692	0.250000	0.0	0.294393	0.0	0.0	0.0	1.0	
4	0.351648	0.218750	0.0	0.266355	0.0	0.0	0.0	1.0	

Now splitting test and train for dataset after dropping features

In [131]:

```
#Splitting train and test sets
target_name = 'left'
X3 = drop_department.drop('left', axis=1)
#now X2 has binned dataframe with no left and no Department col

y3=drop_department[target_name]

X3_train, X3_test, y3_train, y3_test = train_test_split(X3,y3,test_size=0.15, random_state=42, stra
tify=y)

X3_train.head()
```

Out[131]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	Work_accident	Promotion	salary	number_project_cat_1
1611	0.000000	0.640625	0.8	0.869159	0.0	0.0	0.5	0.0
8823	0.307692	0.718750	0.2	0.481308	0.0	0.0	0.5	0.0
2383	0.670330	0.421875	0.2	0.275701	1.0	0.0	0.0	0.0
11306	0.714286	1.000000	0.4	0.714953	0.0	0.0	0.0	0.0
10612	0.395604	0.296875	0.4	0.514019	0.0	0.0	0.0	0.0

Cross Validating to check over fitting and estimate the skill of the model on new dataset, generally have a lower bias than other methods

In [132]:

```
#cross validating
kfold = KFold(n_splits=5, random_state=99)
model = LogisticRegression(C=5)
```

```
results = cross_val_score(model ,X3_train, y3_train, cv=kfold)
results
```

Out[132]:

```
array([0.90265866, 0.91123499, 0.89451115, 0.89922813, 0.90604891])
```

In [133]:

```
model = LogisticRegression(random_state = 99)
model.fit(X3_train,y3_train)
predictions = model.predict(X3_test)
print(classification_report(y3_test,predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.92      0.93      1554
         1.0       0.78      0.84      0.81       504

    accuracy                           0.90      2058
   macro avg       0.86      0.88      0.87      2058
weighted avg       0.91      0.90      0.90      2058
```

Droping features that are not much revelant does'nt improve our model much, its the same , lets try some other filtering methods

Chi squared Test

Does it make sense to remove some features? lets see

Using Chi -square because of its robustness with respect to distribution of data

In [134]:

```
from math import log2
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import SelectKBest, chi2
from scipy import stats
```

In [135]:

```
#Splitting test and train test

target_name = 'left'
X_new = data.drop('left', axis=1)


y_new=data[target_name]

X_new_train, X_new_test, y_new_train, y_new_test = train_test_split(X_new,y_new,test_size=0.15, ran
dom_state=1, stratify=y)

X_new_train.head()
```

Out[135]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	Departm
746	0.296703	0.218750	0.0	0.168224	0.125	0.0	0.0	
8691	0.439560	0.328125	0.2	0.285047	0.125	0.0	0.0	
9865	0.956044	0.234375	0.6	0.509346	0.000	1.0	0.0	
2253	0.142857	0.250000	0.8	0.369159	0.250	0.0	0.0	
6944	0.857143	0.187500	0.4	0.172897	0.000	0.0	0.0	

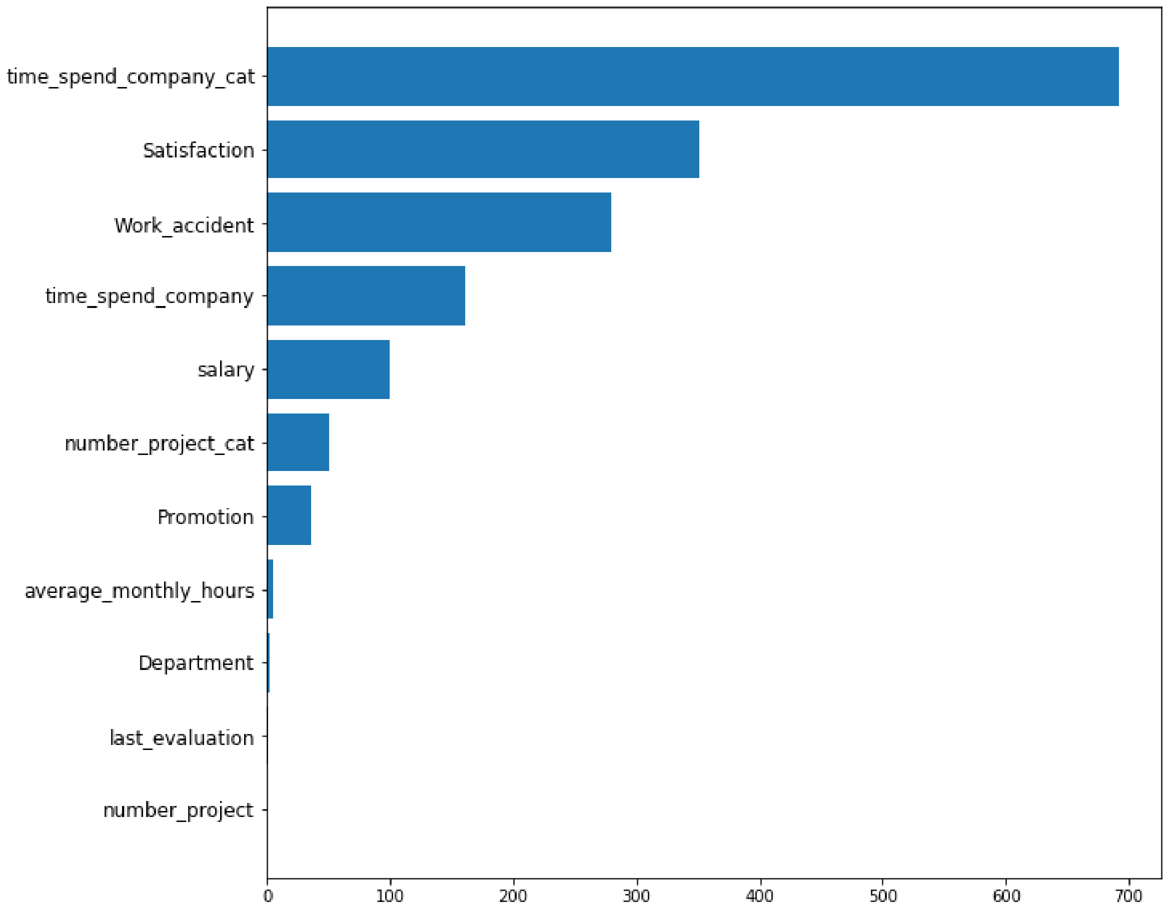
In [136]:

```
chi2_Kbest = SelectKBest(chi2).fit(X_new,y_new)
```

```
#Sorting indices
indices = np.argsort(chi2_Kbest.scores_)[::-1]
#Appending all features
chi2_features = []
for i in range(len(X_new.columns)):
    chi2_features.append(X_new.columns[indices[i]])
```

In [137]:

```
plt.figure(figsize=(10,10))
plt.barh(chi2_features, chi2_Kbest.scores_[indices[range(len(X_new.columns))]])
plt.gca().invert_yaxis()
plt.yticks(fontsize=12)
plt.show()
```



In [138]:

```
data.columns
```

Out[138]:

```
Index(['Satisfaction', 'last_evaluation', 'number_project',
       'average_monthly_hours', 'time_spend_company', 'Work_accident', 'left',
       'Promotion', 'Department', 'salary', 'number_project_cat',
       'time_spend_company_cat'],
      dtype='object')
```

Top seven features are important according to chi squared test,so we will keep top seven features and will remove the rest

In [139]:

```
chi2_features[:7]
```

Out[139]:

```
['time_spend_company_cat',
 'Satisfaction',
```

```
 'Work_accident',
 'time_spend_company',
 'salary',
 'number_project_cat',
 'Promotion']
```

Now will drop all the features which according to chi squared test are not important

In [140]:

```
drop_chi = data.drop(axis =0,columns =[  'last_evaluation', 'number_project',
       'average_monthly_hours',
        'Department'])
drop_chi
```

Out[140]:

	Satisfaction	time_spend_company	Work_accident	left	Promotion	salary	number_project_cat	time_spend_company_cat
0	0.318681	0.125	0.0	1.0	0.0	0.0	1	2
2	0.021978	0.250	0.0	1.0	0.0	0.5	3	3
3	0.692308	0.375	0.0	1.0	0.0	0.0	3	4
4	0.307692	0.125	0.0	1.0	0.0	0.0	1	2
5	0.351648	0.125	0.0	1.0	0.0	0.0	1	2
...	...	...	...	...	...	...	...	...
14994	0.340659	0.125	0.0	1.0	0.0	0.0	1	2
14995	0.307692	0.125	0.0	1.0	0.0	0.0	1	2
14996	0.307692	0.125	0.0	1.0	0.0	0.0	1	2
14997	0.021978	0.250	0.0	1.0	0.0	0.0	3	3
14998	0.307692	0.125	0.0	1.0	0.0	0.0	1	2

13717 rows × 8 columns

Splitting train and test sets

In [141]:

```
target_name = 'left'
X4 = drop_chi.drop('left', axis=1)
y4=drop_chi[target_name]

X4_train, X4_test, y4_train, y4_test = train_test_split(X4,y4,test_size=0.15, random_state=42, stra
tify=y)

X4_train.head()
```

Out[141]:

	Satisfaction	time_spend_company	Work_accident	Promotion	salary	number_project_cat	time_spend_company_cat
1701	0.000000	0.375	0.0	0.0	0.5	3	4
9239	0.307692	0.375	0.0	0.0	0.5	2	4
2514	0.670330	0.250	1.0	0.0	0.0	2	3
12138	0.714286	0.375	0.0	0.0	0.0	2	4
11117	0.395604	0.000	0.0	0.0	0.0	2	1

Cross Validation to prevent over fitting again

In [142]:

```
kfold = KFold(n_splits=5, random_state=99)
model = LogisticRegression(C=5)
results = cross_val_score(model ,X4_train, y4_train, cv=kfold)
```

```
results = cross_val_score(model ,X4_train, y4_train, cv=kfold)
results
```

Out[142]:

```
array([0.86363636, 0.88078902, 0.86620926, 0.86406518, 0.86443586])
```

In [145]:

```
model = LogisticRegression(random_state = 99)
model.fit(X4_train,y4_train)
predictions = model.predict(X4_test)
print(classification_report(y4_test,predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.89      0.90      1554
         1.0       0.68      0.71      0.69       504

    accuracy                           0.85      2058
   macro avg       0.79      0.80      0.80      2058
weighted avg       0.85      0.85      0.85      2058
```

Looks like removing features is not helping to improve our model, so we will drop this idea, will try with Wrapper method once

Wrapper Method

In [148]:

```
%matplotlib inline
from sklearn.feature_selection import RFE
from sklearn.linear_model import RidgeCV, LassoCV, Ridge, Lasso
import statsmodels.api as sm
from statsmodels.formula.api import ols
import statsmodels.formula.api as smf
import numpy as np







import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
%matplotlib inline
from sklearn.model_selection import train_test_split
```

In [149]:

```
target_name = 'left'
X_wrap = data.drop('left', axis=1)
y_wrap=data[target_name]

X_wrap_train, X_wrap_test, y_wrap_train, y_wrap_test = train_test_split(X_wrap,y_wrap,test_size=0.1
5, random_state=42, stratify=y)

X_wrap_train.head()
```

Out[149]:

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	Departr
1701	0.000000	0.640625	0.8	0.869159	0.375	0.0	0.0	
9239	0.307692	0.718750	0.2	0.481308	0.375	0.0	0.0	
2514	0.670330	0.421875	0.2	0.275701	0.250	1.0	0.0	

	Satisfaction	last_evaluation	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	Departr
12138	0.714286	1.000000	0.4	0.714953	0.375	0.0	0.0	
11117	0.395604	0.296875	0.4	0.514019	0.000	0.0	0.0	

Will detect the interaction between variables, maybe we will find the optimal feature subset.

In Wrapper method we will use Backward Elimination, if pvalue is above 0.05 then we will remove the feature

Using OLS model which stands for “Ordinary Least Squares”.

In [150]:

```
#Mandatory for sm.OLS model, adding constant column of one's
import statsmodels.api as sm
X_1 = sm.add_constant(X_wrap)
model = sm.OLS(y_wrap,X_1.astype(float)).fit()
model.pvalues
```

Out[150]:

```
const                     0.000000e+00
Satisfaction              0.000000e+00
last_evaluation           3.617761e-01
number_project            5.972886e-88
average_monthly_hours     2.476909e-10
time_spend_company       4.183335e-228
Work_accident             7.262662e-51
Promotion                 7.478545e-05
Department                4.267991e-01
salary                    1.706206e-39
number_project_cat       9.583172e-213
time_spend_company_cat    0.000000e+00
dtype: float64
```

With the help of loop we will remove all the features whose p-value is above 0.05 and build the model once again.

Took help from https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b

In [151]:

```
#Backward Elimination
cols = list(X_wrap.columns)
pmax = 1
while (len(cols)>0):
    p= []
    X_1 = X_wrap[cols]
    X_1 = sm.add_constant(X_1)
    model = sm.OLS(y_wrap,X_1.astype(float)).fit()
    p = pd.Series(model.pvalues.values[1:],index = cols)
    pmax = max(p)
    feature_with_p_max = p.idxmax()
    if(pmax>0.05):
        cols.remove(feature_with_p_max)
    else:
        break
selected_features_BE = cols
print(selected_features_BE)
```

```
['Satisfaction', 'number_project', 'average_monthly_hours', 'time_spend_company', 'Work_accident',
'Promotion', 'salary', 'number_project_cat', 'time_spend_company_cat']
```

These are the final set of variables

In [152]:

```
# showing features
selected_features_BE[:9]
```

Out[152]:

```
['Satisfaction',
```

```
'number_project',
'average_monthly_hours',
'time_spend_company',
'Work_accident',
'Promotion',
'salary',
'number_project_cat',
'time_spend_company_cat']
```

We will drop those features whose p-value is above 0.05 and will store them in wrap_drop variable

In [153]:

```
wrap_drop =data.drop(axis =0,columns = [ 'last_evaluation',
    'Department' ])
wrap_drop
```

Out[153]:

	Satisfaction	number_project	average_monthly_hours	time_spend_company	Work_accident	left	Promotion	salary	number_pr
0	0.318681	0.0	0.285047	0.125	0.0	1.0	0.0	0.0	
2	0.021978	1.0	0.822430	0.250	0.0	1.0	0.0	0.5	
3	0.692308	0.6	0.593458	0.375	0.0	1.0	0.0	0.0	
4	0.307692	0.0	0.294393	0.125	0.0	1.0	0.0	0.0	
5	0.351648	0.0	0.266355	0.125	0.0	1.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
14994	0.340659	0.0	0.257009	0.125	0.0	1.0	0.0	0.0	
14995	0.307692	0.0	0.299065	0.125	0.0	1.0	0.0	0.0	
14996	0.307692	0.0	0.219626	0.125	0.0	1.0	0.0	0.0	
14997	0.021978	0.8	0.859813	0.250	0.0	1.0	0.0	0.0	
14998	0.307692	0.0	0.289720	0.125	0.0	1.0	0.0	0.0	

13717 rows × 10 columns

Splitting test and train sets

In [154]:

```
target_name = 'left'
X5 = wrap_drop.drop('left', axis=1)
y5= wrap_drop[target_name]

X5_train, X5_test, y5_train, y5_test = train_test_split(X5,y5,test_size=0.15, random_state=42, stra
tify=y)

X5_train.head()
```

Out[154]:

	Satisfaction	number_project	average_monthly_hours	time_spend_company	Work_accident	Promotion	salary	number_project_
1701	0.000000	0.8	0.869159	0.375	0.0	0.0	0.5	
9239	0.307692	0.2	0.481308	0.375	0.0	0.0	0.5	
2514	0.670330	0.2	0.275701	0.250	1.0	0.0	0.0	
12138	0.714286	0.4	0.714953	0.375	0.0	0.0	0.0	
11117	0.395604	0.4	0.514019	0.000	0.0	0.0	0.0	

Cross Validation to prevent over-fitting

In [155]:

```
kfold = KFold(n_splits=5, random_state=99)
```

```
kfold = KFold(n_splits=5, random_state=99)
model = LogisticRegression(C=5)
results = cross_val_score(model ,X5_train, y5_train, cv=kfold)
results
```

Out[155]:

```
array([0.87135506, 0.88379074, 0.87178388, 0.86578045, 0.87773488])
```

In [157]:

```
model =linear_model.LogisticRegression(random_state = 99)
model.fit(X5_train,y5_train)
predictions = model.predict(X5_test)
print(classification_report(y5_test,predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.90      0.90      1554
         1.0       0.70      0.70      0.70       504

    accuracy                           0.85      2058
   macro avg       0.80      0.80      0.80      2058
weighted avg       0.85      0.85      0.85      2058
```

Accuracy of our model is the same as before , looks like filtering methods are not helping much, lets try something else

Polynomial Features

Let's construct new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to two. I am using polynomial features beacuse it provides the best approximation of the relationship between the dependent and indepenent variable.

To overcome under-fitting, sometimes we need to increase the complixity of the model.

A simple model may suffer from high bias (underfitting), while a complex model may suffer from high variance (overfitting) leading to a bias-variance trade-off.

In [158]:

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(2)
poly_features = poly.fit_transform(data.drop('left', axis=1))
```

Checking the shape of our new features

In [159]:

```
poly_features.shape
```

Out[159]:

```
(13717, 78)
```

Splitting dataset into test and train, as poly_features does not contain target variable therefore i took it from previous dataset

In [160]:

```
target_name = 'left'
X6 = poly_features


y6=data[target_name]

X6_train, X6_test, y6_train, y6_test = train_test_split(X6,y6,test_size=0.15, random_state=42, stra
tify=y)
```

```
X6_train
```

Out[160]:

```
array([[ 1.        ,  0.        ,  0.640625  , ...,  9.        ,
        12.        , 16.        ],
       [ 1.        ,  0.30769231,  0.71875   , ...,  4.        ,
         8.        , 16.        ],
       [ 1.        ,  0.67032967,  0.421875  , ...,  4.        ,
         6.        ,  9.        ],
       ...,
       [ 1.        ,  0.75824176,  0.0625    , ...,  4.        ,
         8.        , 16.        ],
       [ 1.        ,  1.        ,  0.3125    , ...,  9.        ,
         6.        ,  4.        ],
       [ 1.        ,  0.36263736,  0.28125   , ...,  4.        ,
         6.        ,  9.        ]])
```

Cross Validation to prevent over-fitting

In [161]:

```
kfold = KFold(n_splits=5, random_state=99)

results = cross_val_score(model ,X6_train, y6_train, cv=kfold)
results
```

Out[161]:

```
array([0.94468268, 0.94468268, 0.95283019, 0.92924528, 0.95323895])
```

In [77]:

```
model=LogisticRegression(C=7,penalty='l2')
model.fit(X6_train, y6_train)
predictions = model.predict(X6_test)
print(classification_report(y6_test,predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.96      0.97      1554
         1.0       0.88      0.91      0.89       504

    accuracy                           0.95      2058
   macro avg       0.93      0.94      0.93      2058
weighted avg       0.95      0.95      0.95      2058
```

Creating combination of features really improved our model, because there is every possible combination of features now as the degree is 2 ,polynomial features are (1,a,b,a^2,ab,b^2).

Performance is quite better, we have an accuracy of 95% , got better in predicting '0' (a employee is NOT going to leave the company) than '1' (a employee is going to leave the company)