Hi everyone.

This notebook is part of a project developed fot the Landing.Jobs Data Challenge

If you want to read more about it, you can read the abstract and the presentation through these links:

Abstract: https://drive.google.com/file/d/1qeOC-0TPkGPZHNiGgIaj6g-cPKNCuAP_/view?usp=sharing (https://drive.google.com/file/d/1qeOC-0TPkGPZHNiGgIaj6g-cPKNCuAP_/view?usp=sharing)

Presentation: https://drive.google.com/file/d/1eCMqhjT_xGBNw_JSAkUrW_FgkNSMtWIv/view?usp=sharing (https://drive.google.com/file/d/1eCMqhjT_xGBNw_JSAkUrW_FgkNSMtWIv/view?usp=sharing)

Goal: Developing a model able to predict if a tech professional earns more than 30k€. This model aims to be ethical and will help anyone that wants to ask for a raise, regardless the gender, nationality and age.

To start, let's import all the libraries that will help us to build the model and do data transformation

In [1]:

```
import pandas as pd
import numpy as np
from numpy import mean
from numpy import isnan
from numpy import asarray
from numpy import polyfit
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
```

Brief Data Analysis

- I really recommend to read the Lanfing.Jobs Report 2021. A lot of decisions were made based on the report analysis.

We need to undestand the data we are working with. In this section I will delete irrelevant columns to solve this problem, due to not not presenting any value or because they may have duplicated information in some way.

In [2]:

```
df = pd.read_csv('LJDV.csv', sep = ';', )
df.index = df.ID
df.shape
```

```
(3371, 126)
```

finding columns that might be irrelevant, having only one unique value

In [3]:

```
# All Language and Framework columns are expected to have only 1.
#But some have 0 and I'll drop those as well
df.nunique()[df.nunique() < 2]
```

```
Residence_Country                 1
Language_JavaScript               1
Language_Bash/Shell/PowerShell    1
Language_SQL                      1
Language_Java                     1
Language_C#                       1
Language_Python                   1
Language_PHP                      1
Language_C++                      1
Language_C                        1
Language_TypeScript               1
Language_Ruby                     1
Language_Swift                    1
Language_Objective-C              1
Language_VB.NET                   1
Language_Assembly                 1
Language_R                        1
Language_Perl                     1
Language_VBA                      1
Language_Matlab                   1
Language_Go                       1
Language_Scala                    1
Language_Groovy                   1
Language_Coffee Script            1
Language_Visual Basic 6           1
Language_Lua                      1
Language_Haskell                  1
Language_HTML/CSS                 1
Language_Kotlin                   1
Language_Rust                     1
Language_Elixir                   1
Language_Clojure                  1
Language_WebAssembly              1
Language_Dart                     1
Language_Languages_N/A            0
Framework_jQuery                  1
Framework_.NET                    1
Framework_Angular/Angular.js      1
Framework_Ruby on Rails           1
Framework_React                   1
Framework_Django                  1
Framework_Laravel                 1
Framework_Spring                  1
Framework_Vue.js                  1
Framework_Express                 1
Framework_Meteor                  1
Framework_Flask                   1
Framework_Ember.js                1
Framework_Drupal                  1
Framework_OutSystems              1
Framework_Framework_N/A           0
dtype: int64
```

In [4]:

```
df = df.drop(['ID','Work_as_Contractor_12m','Work_as_Perm','Remote_Working_Current','Remote_Working_Current_Flexible_Office_Days','Remote_Working_due_to_Covid','Remote_Work_Opinion', 'Residence_Country','Language_Languages_N/A','Framework_Framework_N/A','Employment_Status_Aggregated','Residence_District_Aggregated','Work_Company_Country','Work_Company_Continent','Work_Company_PT_District_Aggregated','Remote_Working_Current_Flexible_Office_Days','Job_Role_Original','Job_Role_Other','Employer_Industry_Other','Employer_Org_Type_Other','Contractor_Avg_Project','Contractor_Avg_Annual_Salary','Perm_GAS_Avg','Perm_GAS_Low_Limit','Perm_GAS_High_Limit','Avg_Salary','Salary_Change'], axis=1)
df = df.drop(['Salary_Fairness','Changing_Jobs_next_6_months','Job_Motivator_Work_life_balance', 'Job_Motivator_Compensation_and_benefits','Job_Motivator_Training/Development_programs_at_work', 'Job_Motivator_Career_growth_opportunities','Job_Motivator_Remote_working','Job_Motivator_Flexible_schedule','Job_Motivator_Company_culture',"Job_Motivator_The_technologies_I'm_working_with", 'Job_Motivator_Versatility/Variety_of_projects','Job_Motivator_Freedom_to_choose_the_clients_and/or_projects','Job_Motivator_Being_autonomous_at_work','Job_Motivator_How_widely_used_or_impactful_the_product/service_I_work_on_is','Job_Motivator_Environmentally_friendly/responsible_work_practice','Job_Perk_Meals_allowance/Company_provided_meals_or_snacks','Job_Perk_Transportation_benefit','Job_Perk_Health_benefits','Job_Perk_Fitness_or_wellness_benefit_(ex._gym_membership)','Job_Perk_Computer/_Office_equipment_allowance','Job_Perk_Professional_development_sponsorship','Job_Perk_Annual_bonus','Job_Perk_Long-term_leave','Job_Perk_Parental_leave','Job_Perk_Stock_options_or_shares','Job_Perk_Education_sponsorship','Job_Perk_Child_care'], axis=1)
df = df.drop(['Birth_Year','Way_Into_Tech_Other','Working_Experience_Aggregated'], axis=1)
df = df.drop(['Language_Languages_Other','Framework_Framework_Other'], axis=1)
df = df.drop(['Contractor_Avg_Project_Intervals','Contractor_Avg_Hour_Rate'], axis=1)
```

From now on, I'll work with a dataset only with Full time employees

In [5]:

```
df_ft = df[df['Employment_Status'] == 'Employed full-time'].drop(['Employment_Status'], axis = 1)
```

In [6]:

```
df_ft.nunique()[df_ft.nunique() < 1]
```

```
Series([], dtype: int64)
```

Data Transformation

Due to the nature of the features, we have a lot of nan values. For the model to process the data, I'll convert all nan values to 0 or even remove some rows, that can create samples with incomplete information. The Target variable will be the "Per_GAS". Below 30k will be 0, Above 30k will be 1. In this way, I create a binary classification problem.

Our dataset has 2931 rows right now. The Target variable will be divided with 1669 samples that are below 30k, target '0', and 1262 samples that are above 30k, target '1'.

- 1: 43.06%
- 0: 56.94%

Categorical Columns - Some columns have a hierarchical value and for those cases I'll use numeric labeling to keep their value. In other cases, those columns are purely categorical and the solution will be one hot encoding

In the end will also change the data type of all the dataset to float and normalize the data

A little disclaimer: Haven't removed any outlier, because removing them will harm the model. I've tried the isolation forest and z-score methods.

In [7]:

```
df_ft['Work_Company_PT_District'].replace(np.nan, 0,inplace=True)
```

In [8]:

```
df_ft.isnull().sum()[df_ft.isnull().sum() > 0]
```

```
Employer_Industry                  102
Employer_Org_Type                   30
Language_JavaScript               1541
Language_Bash/Shell/PowerShell    2564
Language_SQL                      1670
Language_Java                     2377
Language_C#                       2428
Language_Python                   2565
Language_PHP                      2708
Language_C++                      2976
Language_C                        3040
Language_TypeScript               2473
Language_Ruby                     2988
Language_Swift                    3028
Language_Objective-C              3080
Language_VB.NET                   3027
Language_Assembly                 3120
Language_R                        3072
Language_Perl                     3104
Language_VBA                      3038
Language_Matlab                   3111
Language_Go                       3011
Language_Scala                    3065
Language_Groovy                   3057
Language_Coffee Script            3122
Language_Visual Basic 6           3089
Language_Lua                      3114
Language_Haskell                  3129
Language_HTML/CSS                 2014
Language_Kotlin                   2961
Language_Rust                     3109
Language_Elixir                   3100
Language_Clojure                  3124
Language_WebAssembly              3116
Language_Dart                     3090
Framework_jQuery                  2410
Framework_.NET                    2404
Framework_Angular/Angular.js      2598
Framework_Ruby on Rails           2998
Framework_React                   2447
Framework_Django                  3004
Framework_Laravel                 2954
Framework_Spring                  2745
Framework_Vue.js                  2870
Framework_Express                 2898
Framework_Meteor                  3126
Framework_Flask                   3031
Framework_Ember.js                3115
Framework_Drupal                  3088
Framework_OutSystems              3083
Age                                  4
Way_Into_Tech                       69
Education_Level                      2
dtype: int64
```

In [9]:

```
df_ft = df_ft[df_ft['Employer_Industry'].notna()]
df_ft = df_ft[df_ft['Employer_Org_Type'].notna()]
df_ft = df_ft[df_ft['Age'].notna()]
df_ft = df_ft[df_ft['Way_Into_Tech'].notna()]
df_ft = df_ft[df_ft['Education_Level'].notna()]
```

In [10]:

```
columns_0 = df_ft.isnull().sum()[df_ft.isnull().sum() > 0].index
df_ft = df_ft.fillna(0)
df_help = df_ft.copy()
df_ft
```

	Residence_District	Work_Company_PT_International	Work_Company_PT_District	Job_Remote_or_Office	Job_Role
ID					
4	Açores	Portugal	Açores	Full Office Job	Product Owner/Product Manager
5	Braga	Portugal	Braga	Remote Job (full or flexible)	Back-End Developer
7	Aveiro	Portugal	Aveiro	Full Office Job	Front-End Developer
8	Santarém	Portugal	Lisboa	Full Office Job	Full-Stack Developer
9	Viseu	Portugal	Viseu	Full Office Job	Full-Stack Developer
...	...	...	...	...	...
3365	Lisboa	International	0	Remote Job (full or flexible)	Maintenance & Support
3367	Porto	Portugal	Porto	Remote Job (full or flexible)	Full-Stack Developer
3368	Lisboa	Portugal	Lisboa	Full Office Job	Quality Assurance/Testing
3369	Porto	International	0	Full Office Job	Front-End Developer
3370	Porto	Portugal	Porto	Remote Job (full or flexible)	Front-End Developer

2931 rows × 65 columns

Transforming Perm_GAS as Target with numeric values

In [11]:

```
def f(row):
    if row['Perm_GAS'] == '< €15.000':
        val = 0
    if row['Perm_GAS'] == '€15.000 - €20.000':
        val = 0
    if row['Perm_GAS'] == '€20.000 - €25.000':
        val = 0
    if row['Perm_GAS'] == '€25.000 - €30.000':
        val = 0
    if row['Perm_GAS'] == '€30.000 - €35.000':
        val = 1
    if row['Perm_GAS'] == '€35.000 - €40.000':
        val = 1
    if row['Perm_GAS'] == '€40.000 - €45.000':
        val = 1
    if row['Perm_GAS'] == '€45.000 - €50.000':
        val = 1
    if row['Perm_GAS'] == '€50.000 - €55.000':
        val = 1
    if row['Perm_GAS'] == '€55.000 - €60.000':
        val = 1
    if row['Perm_GAS'] == '€60.000 - €65.000':
        val = 1
    if row['Perm_GAS'] == '€65.000 - €70.000':
        val = 1
    if row['Perm_GAS'] == '€70.000 - €75.000':
        val = 1
    if row['Perm_GAS'] == '€75.000 - €80.000':
        val = 1
    if row['Perm_GAS'] == '€80.000 - €85.000':
        val = 1
    if row['Perm_GAS'] == '€85.000 - €90.000':
        val = 1
    if row['Perm_GAS'] == '€90.000 - €95.000':
        val = 1
    if row['Perm_GAS'] == '€95.000 - €100.000':
        val = 1
    if row['Perm_GAS'] == '> €100.000':
        val = 1
    return val
```

In [12]:

```
df_ft['Target'] = df_ft.apply(f, axis=1)
df_ft = df_ft.drop(['Perm_GAS'], axis = 1)
df_ft['Target'].value_counts().sort_values()
```

```
1    1262
0    1669
Name: Target, dtype: int64
```

In [13]:

```
df_ft.shape
```

```
(2931, 65)
```

In [14]:

```
df_ft.Target.isnull().sum()
```

```
0
```

Solving categorical colummns Problem

creating array with categorical columns

In [15]:

```
cat_col = ['Residence_District','Work_Company_PT_District','Work_Company_PT_International','Job_Remote_or_Office','Job_Role','Employer_Industry','Employer_Org_Type','Employer_Size','Way_Into_Tech','Education_Level','Perm_Current_Company_how_long','Citizenship','Gender','English_Level','Working_Experience']
```

Creating Dummies for each categorical column

In [16]:

```
#Creating new columns for each element
encoded_residence = pd.get_dummies(df_ft['Residence_District'], prefix = 'Colab_Resid')
#joining the results to our dataset
df_ft = df_ft.join(encoded_residence).drop('Residence_District', axis=1)
```

In [17]:

```
#Creating new columns for each element
Work_Company_PT_District = pd.get_dummies(df_ft['Work_Company_PT_District'], prefix = 'Comp_District')
#joining the results to our dataset
df_ft = df_ft.join(Work_Company_PT_District).drop('Work_Company_PT_District', axis=1)
```

In [18]:

```
#Creating new columns for each element
Work_Company_PT_International = pd.get_dummies(df_ft['Work_Company_PT_International'], prefix = 'Comp_Country')
#joining the results to our dataset
df_ft = df_ft.join(Work_Company_PT_International).drop('Work_Company_PT_International', axis=1)
```

In [19]:

```
#Creating new columns for each element
Job_Remote_or_Office = pd.get_dummies(df_ft['Job_Remote_or_Office'], prefix = 'Job_rem_off')
#joining the results to our dataset
df_ft = df_ft.join(Job_Remote_or_Office).drop('Job_Remote_or_Office', axis=1)
```

In [20]:

```
#Creating new columns for each element
Job_Role = pd.get_dummies(df_ft['Job_Role'], prefix = 'job_role')
#joining the results to our dataset
df_ft = df_ft.join(Job_Role).drop('Job_Role', axis=1)
```

In [21]:

```
#Creating new columns for each element
Employer_Industry = pd.get_dummies(df_ft['Employer_Industry'], prefix = 'Employer_Industry')
#joining the results to our dataset
df_ft = df_ft.join(Employer_Industry).drop('Employer_Industry', axis=1)
```

In [22]:

```
#Creating new columns for each element

Employer_Org_Type = pd.get_dummies(df_ft['Employer_Org_Type'], prefix = 'Type')

#there is a column with a character "<", this character creates an error in some models. Will re
move it:
Employer_Org_Type['Type_SME - Small or Medium Enterprise (personnel -250)'] = Employer_Org_Type[
'Type_SME - Small or Medium Enterprise (personnel <250)']
Employer_Org_Type = Employer_Org_Type.drop(['Type_SME - Small or Medium Enterprise (personnel <2
50)'], axis = 1)

#joining the results to our dataset
df_ft = df_ft.join(Employer_Org_Type).drop('Employer_Org_Type', axis=1)
```

In [23]:

```
#In this case, for "Employer Size", because we can translate this variable into numerical value,
I will attribute a label for each size level

df_ft['Employer_Size'] = df_ft['Employer_Size'].map({'Less than 10 employees': 0, '10 - 19 emplo
yees': 1,
                                    '20 - 99 employees': 2, '100 - 499 employees': 3,
                                    '500 - 999 employees': 4, '1000 - 4.999 employees': 5, 'More
than 5.000 employees': 6})
```

In [24]:

```
#Creating new columns for each element
Way_Into_Tech = pd.get_dummies(df_ft['Way_Into_Tech'], prefix = 'Way_Into_Tech')
#joining the results to our dataset
df_ft = df_ft.join(Way_Into_Tech).drop('Way_Into_Tech', axis=1)
```

In [25]:

```
#In this case, for "Education_Level", because we can translate this variable into numerical valu
e, I will attribute a label for each size level

df_ft['Education_Level'] = df_ft['Education_Level'].map({'I prefer not to answer': 0, 'Basic Edu
cation': 1,
                                    'High School Education': 2, 'Trade/technical/vocational trai
ning': 3,
                                    'University drop out': 4, 'Bachelor degree': 5, 'Masters deg
ree': 6, 'Doctoral degree': 7})
```

In [26]:

```
#In this case, for "Perm_Current_Company_how_long", because we can translate this variable into
 numerical value, I will attribute a label for each size level

df_ft['Perm_Current_Company_how_long'] = df_ft['Perm_Current_Company_how_long'].map({'Less than
 one year': 0, 'Between 1 - 3 years': 1,
                                    'More than 3 years': 2})
```

In [27]:

```
#Beacause of the ethics associated with this model, this feature will be removed.
df_ft = df_ft.drop('Citizenship', axis=1)
```

In [28]:

```
#Beacause of the ethics associated with this model, this feature will be removed.
df_ft = df_ft.drop('Gender', axis=1)
```

In [29]:

```
#In this case, for "English_Level", because we can translate this variable into numerical value,
I will attribute a label for each size level

df_ft['English_Level'] = df_ft['English_Level'].map({'Elementary': 0, 'Limited working proficien
cy': 1,
                                    'Professional working proficiency': 2, 'Full professional pr
oficiency': 3, 'Native or bilingual proficiency': 4})
```

In [30]:

```
#In this case, for "Working_Experience", because we can translate this variable into numerical v
alue, I will attribute a label for each size level

df_ft['Working_Experience'] = df_ft['Working_Experience'].map({'No working experience': 0, 'Less
than 1 year': 1,
                                    'Between 1 - 3 years': 2, 'Between 3 - 6 years': 3, 'Between
6 - 9 years': 4, 'More than 9 years': 5})
```

In [31]:

```
#Beacause of the ethics associated with this model, this feature will be removed.
df_ft = df_ft.drop('Age', axis=1)
```

In [32]:

```
#defining all string as 1 in languages and frameworks columns

#df_help1 = df_help.iloc[:, 8:56]

df_ft['Employer_Size_'] = df_ft['Employer_Size'].copy()
df_ft = df_ft.drop(['Employer_Size'], axis = 1)
#creating a new dataset with language and framework to turn them to numeric
df_help1 = df_ft.iloc[:, 0:48]
#New column with the sum of how many languages and frameworks does one knows
df_help1[df_help1 != 0] = 1
df_ft = df_ft.iloc[:, 48:154]
#join both datasets again into one
df_ft = df_ft.join(df_help1)
df_ft.shape
```

```
(2931, 152)
```

Changing data Types

In [33]:

```
df_ft = df_ft.astype(float)
```

Dropping Unnecessary columns

In [34]:

```
#when the company is international
df_ft['Comp_District_0'].sum()

#repeated informatino in columns that tell us if it is remote or not
df_ft = df_ft.drop(['Comp_District_0','Comp_Country_Portugal','Job_rem_off_Remote Job (full or f
lexible)'], axis = 1)
```

In [35]:

```
df_ft
```

	Perm_Current_Company_how_long	English_Level	Education_Level	Working_Experience	Target	Colab_Resid_Aveiro	Co
ID							
4	1.0	3.0	5.0	5.0	0.0	0.0	1.(
5	2.0	2.0	5.0	4.0	0.0	0.0	0.(
7	1.0	3.0	5.0	3.0	0.0	1.0	0.(
8	2.0	4.0	5.0	5.0	0.0	0.0	0.(
9	1.0	1.0	5.0	2.0	0.0	0.0	0.(
...	...	...	...	...	...	...	...
3365	1.0	3.0	4.0	5.0	1.0	0.0	0.(
3367	1.0	3.0	4.0	3.0	0.0	0.0	0.(
3368	1.0	2.0	4.0	3.0	0.0	0.0	0.(
3369	2.0	2.0	4.0	4.0	1.0	0.0	0.(
3370	1.0	3.0	4.0	3.0	0.0	0.0	0.(

2931 rows × 149 columns

Normalizing Data

In [36]:

```
scaler = MinMaxScaler()
#creating new variable with target column to keep as discrete and not continuous
y = df_ft['Target'].astype(int)
df_ft = df_ft.drop(['Target'], axis = 1)
df_ft = pd.DataFrame(scaler.fit_transform(df_ft), columns=df_ft.columns)
df_ft['Target'] = y.reset_index().drop(['ID'], axis = 1).Target
y = y.reset_index().drop(['ID'], axis = 1).Target
```

In [37]:

```
X = df_ft.drop(['Target'], axis = 1)
```

Creating Training and Test set

For this model, will be create a Train and validation test. The final score (accuracy) will be validated with cross validation.

In [38]:

```
#Creating test and train dataset
x_train, x_val, y_train, y_val = train_test_split(X, y, test_size=0.20, random_state=0)

x_train = x_train.reset_index().drop(['index'], axis = 1)
x_val = x_val.reset_index().drop(['index'], axis = 1)

y_train = y_train.reset_index().Target
y_val = y_val.reset_index().Target
```

In [39]:

```
x_train.shape
```

(2344, 148)

In [40]:

```
x_val.shape
```

(587, 148)

What model to use

In order to know what model to use and to start to understand the data to extract the best accuracy possible, I'll run several models to know how will they perform. To choose the final model, I'll always hav in consideration that I want not just the best model, but the best model with a good level of interpretability from their coefficients.

get a list of models to evaluate

def get_models(): models = list() models.append(LogisticRegression()) models.append(RidgeClassifier()) models.append(SGDClassifier()) models.append(PassiveAggressiveClassifier()) models.append(KNeighborsClassifier()) models.append(DecisionTreeClassifier()) models.append(ExtraTreeClassifier()) models.append(LinearSVC()) models.append(SVC()) models.append(GaussianNB()) models.append(AdaBoostClassifier()) models.append(BaggingClassifier()) models.append(RandomForestClassifier()) models.append(ExtraTreesClassifier()) models.append(GaussianProcessClassifier()) models.append(GradientBoostingClassifier()) models.append(LinearDiscriminantAnalysis()) models.append(QuadraticDiscriminantAnalysis()) return models

def evaluate_model(cv, model): scores = cross_val_score(model, x_train, y_train, scoring='accuracy', cv=cv, n_jobs=-1) return mean(scores)

cv_2 = KFold(n_splits=5, shuffle=True, random_state=1) cv = KFold(n_splits=20, shuffle=True, random_state=1) models = get_models() cv_2_results, cv_results = list(), list() for model in models: cv_mean = evaluate_model(cv, model) cv_2_mean = evaluate_model(cv_2, model) if isnan(cv_mean) or isnan(cv_2_mean): continue cv_results.append(cv_mean) cv_2_results.append(cv_2_mean) print('>%s: cv_2=%.3f, cv=%.3f' % (type(model).**name**, cv_2_mean, cv_mean))

In [ ]:

Most important features / Feature reduction

I'll try to understand what are the right features to integrate in the model. For this case, I'll also bear in mind that PCA is a technic that can possibly improve accuracy but will lack interpretability. The final results will show that PCA results do in fact improve the model, but not so much. For that reason, I'll choose Logistic Regression and the features included will be the number of features of higher coefficient that return the highest accuracy rate.

Top features from PCA on Logistic Regression

In [41]:

```
#I'll try to understand what are the best fetures to optimize the model from PCA
from sklearn.decomposition import PCA

score_list = []
for x in np.arange(1,x_train.shape[1]):
    #Defining the columns set
    pca = PCA(n_components = x)
    pca.fit(x_train)
    x_trainPCA = pca.transform(x_train)
    x_valPCA = pca.transform(x_val)
    #Creating Train and Test
    #Score from LogReg
    logisticRegr = LogisticRegression()
    logisticRegr.fit(x_trainPCA, y_train)
    score = logisticRegr.score(x_valPCA, y_val)
    score_list.append([score,x])
```

In [42]:

```
topPCA = pd.DataFrame(score_list).sort_values(0,ascending = False)
topPCA
```

	0	1
71	0.814310	72
72	0.809199	73
73	0.807496	74
75	0.807496	76
68	0.807496	69
...	...	...
4	0.618399	5
3	0.618399	4
2	0.592845	3
0	0.591141	1
1	0.589438	2

147 rows × 2 columns

In [ ]:

Top features from a LogReg with top correlation matrix

Building the LogReg with all columns and building a dataframe with top coefficients:

In [43]:

```
#Getting top correlated columns with the Target. As expected, because of our coefficients in Log
Reg, the highest value is for "Working Experience" with 0.48.
cor = df_ft.corr()
col3 = cor.sort_values('Target')['Target'].sort_values(ascending = False)
```

In [44]:

```
#I'll try to understand what is the optimal number of top features from the correlation matrix
 (pearson) to get the best accuracy rate.

score_list = []
for x in range(2,149):
    #Defining the columns set
    cols_3 = X[col3.index[1:x]].columns
    x_train_LRcor = x_train[cols_3]
    x_val_LRcor = x_val[cols_3]
    #Creating Train and Test
    #Score from LogReg
    logisticRegr = LogisticRegression()
    logisticRegr.fit(x_train_LRcor, y_train)
    score = logisticRegr.score(x_val_LRcor, y_val)
    score_list.append(score)
```

In [45]:

```
pd.options.display.max_rows = 50
pd.DataFrame(score_list).sort_values(0, ascending = False).head(5)
```

	0
146	0.805792
122	0.800681
124	0.798978
123	0.798978
79	0.797274

Top features from a LogReg with top coef from LogReg with all columns

In [46]:

```
logisticRegr = LogisticRegression(solver='newton-cg')
logisticRegr.fit(x_train, y_train)
score = logisticRegr.score(x_val, y_val)
print(score)

pd.options.display.max_rows = 500

importance = logisticRegr.coef_

cols_importance = pd.DataFrame(importance)
imp = cols_importance.T
imp['columns'] = X.columns
imp = imp.sort_values(by = 0, ascending = False).reset_index().drop(['index'], axis = 1)
```

```
0.8006814310051107
```

In [47]:

```
#I'll try to understand what is the optimal number of top features from the logreg coefficients
 to get the best accuracy rate.

score_list_LR = []

for x in range(1,160):
    #Defining the columns set
    cols_2 = list(imp['columns'][0:x])
    x_train_LR = x_train[cols_2]
    x_val_LR = x_val[cols_2]
    #Score from LogReg
    logisticRegr = LogisticRegression(solver='newton-cg')
    logisticRegr.fit(x_train_LR, y_train)
    score = logisticRegr.score(x_val_LR, y_val)
    score_list_LR.append([score,x])
```

In [48]:

```
pd.options.display.max_rows = 50
top_LR = pd.DataFrame(score_list_LR).sort_values(0, ascending = False).head(5)
top_LR
```

	0	1
139	0.807496	140
140	0.807496	141
141	0.805792	142
144	0.804089	145
143	0.804089	144

Creating Our Model from previous experiences

I've also tried a model with GBoost and XGBoost that would give great results sometimes, but due to their stochastic nature, those good results were very uncertain. When I tried to stabilize the stochasticity of those models, the accuracy fell to levels below Logistic Regression.

The final model will be made of a Logistic Regression for prediction, using top features from logistic regression coefficients. It's simple and very easy to use and I hope I can optimize it a little bit.

Filtering our Train and Test sets with the best results obtained before

In [49]:

```
cols_2 = list(imp['columns'][0:top_LR[1].iloc[0]])
x_train = x_train[cols_2]
x_val = x_val[cols_2]
```

In [50]:

```
x_train.shape
```

```
(2344, 140)
```

Applying the model with the new sets, should give us the best result obtained previously

In [51]:

```
logisticRegr = LogisticRegression(solver='newton-cg')
logisticRegr.fit(x_train, y_train)
score = logisticRegr.score(x_val, y_val)
score
```

```
0.807495741056218
```

This step is very important and very interesting. We are now able to read the coefficients for each feature and understand their importance. Working Experience is clearly the king in our model and completely makes sense: The higher your experience, the higher will be your wage.

In [52]:

```
#Finding the coefficient values for each column

importance = logisticRegr.coef_

cols_importance = pd.DataFrame(importance)
imp = cols_importance.T
imp['columns'] = x_train.columns
imp['absolute'] = imp[0].abs()
imp = imp.sort_values(by = 'absolute', ascending = False).reset_index().drop(['index'], axis = 1
)
imp.head(20)
```

	0	columns	absolute
0	4.915877	Working_Experience	4.915877
1	1.866013	Comp_Country_International	1.866013
2	1.516652	Type_Scale-up (fast growing company aka "unico...	1.516652
3	1.389163	Colab_Resid_Lisboa	1.389163
4	1.348296	Type_Startup (new business venture)	1.348296
5	1.193807	Education_Level	1.193807
6	1.055859	Comp_District_Lisboa	1.055859
7	1.031700	Employer_Size_	1.031700
8	1.024210	Colab_Resid_Setúbal	1.024210
9	1.009382	Employer_Industry_Media, advertising, publishi...	1.009382
10	0.961249	English_Level	0.961249
11	0.950513	job_role_Technical Team Leader	0.950513
12	0.942491	job_role_CTO	0.942491
13	0.910483	Comp_District_Coimbra	0.910483
14	0.809730	Employer_Industry_Transportation	0.809730
15	0.804941	Comp_District_Porto	0.804941
16	0.785593	Employer_Industry_Energy or utilities	0.785593
17	0.774407	job_role_Scrum Master	0.774407
18	0.774021	Language_Perl	0.774021
19	0.753584	Type_Corporate	0.753584

Bagging

From this point, I've tried several ways to improve the model and this one turned out to be a fast way of improving the accuracy a little bit

I've decided to use the top 3 models from the previous results from the process where I've tried several models and create a separated dataset with 3 columns, each one for the prediction of each model. The models used are:

- Logistic Regression
- XGBoost
- SVC

Creating new column with prediction probabilty #1

In [53]:

```
x_train_pred = logisticRegr.predict_proba(x_train)[:,1]
x_val_pred = logisticRegr.predict_proba(x_val)[:,1]

#creating 1st prediction column

train_pred = pd.DataFrame()
val_pred = pd.DataFrame()
test_pred = pd.DataFrame()

train_pred['LR'] = x_train_pred
val_pred['LR'] = x_val_pred
```

Creating new column with prediction probabilty #2 - using XGB

In [54]:

```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


model = XGBClassifier()
model.fit(x_train, y_train)

# make predictions for test data
x_train_pred = model.predict_proba(x_train)[:,1]
x_val_pred = model.predict_proba(x_val)[:,1]


#creating 2nd prediction column

train_pred['XGB'] = x_train_pred
val_pred['XGB'] = x_val_pred
```

Creating new column with prediction probabilty #3 - using SVC

In [55]:

```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

SVC2 = SVC(probability=True)

SVC2.fit(x_train,y_train)

# make predictions for test data
x_train_pred = SVC2.predict_proba(x_train)[:,1]
x_val_pred = SVC2.predict_proba(x_val)[:,1]

#creating 3rd prediction column

train_pred['SVC'] = x_train_pred
val_pred['SVC'] = x_val_pred
```

In [ ]:

Final Model

The final model will be a Logistic regression, but now, with a new feature that will be the average of predictions by the bagging process. For the final model, I'll use cross validation and will perform feature reduction one more time, because we will have a new extra feature that will change the correlation between the data.

Joining our predictions into the datasets

In [56]:

```
logreg = LogisticRegression(solver='newton-cg')
param = {'C':[1,10]}
LR = GridSearchCV(logreg,param,scoring='accuracy',refit=True,cv=10)
LR.fit(x_train,y_train)
score = LR.score(x_val, y_val)
print(LR.best_params_,score)
```

```
{'C': 1} 0.807495741056218
```

In [57]:

```
x_train = pd.DataFrame(x_train)
x_val = pd.DataFrame(x_val)
```

In [58]:

```
x_train = x_train.join(train_pred)
x_val = x_val.join(val_pred)
```

We will add more value joining those 3 columns by their mean

In [59]:

```
x_train['proba'] = (x_train['LR'] + x_train['XGB'] + x_train['SVC']) / 3
x_train = x_train.drop(['LR', 'XGB', 'SVC'], axis = 1)

x_val['proba'] = (x_val['LR'] + x_val['XGB'] + x_val['SVC']) / 3
x_val = x_val.drop(['LR', 'XGB', 'SVC'], axis = 1)
```

Let's check how does our model behaves with this new data

In [60]:

```
logreg = LogisticRegression(solver='newton-cg')
param = {'C':[1,10]}
LR = GridSearchCV(logreg,param,scoring='accuracy',refit=True,cv=10)
LR.fit(x_train,y_train)
score = LR.score(x_val, y_val)
print(LR.best_params_,score)
```

```
{'C': 10} 0.817717206132879
```

Last step: one more feature reduction

In [61]:

```
logisticRegr = LogisticRegression(solver='newton-cg')
logisticRegr.fit(x_train, y_train)
score = logisticRegr.score(x_val, y_val)
print(score)

pd.options.display.max_rows = 500

importance = logisticRegr.coef_

cols_importance = pd.DataFrame(importance)
imp = cols_importance.T
imp['columns'] = x_train.columns
imp = imp.sort_values(by = 0, ascending = False).reset_index().drop(['index'], axis = 1)
```

```
0.817717206132879
```

In [62]:

```
score_list_LR = []

for x in range(1,160):
    #Defining the columns set
    cols_2 = list(imp['columns'][0:x])
    x_train_LR = x_train[cols_2]
    x_val_LR = x_val[cols_2]
    #Score from LogReg
    logisticRegr = LogisticRegression(solver='newton-cg')
    logisticRegr.fit(x_train_LR, y_train)
    score = logisticRegr.score(x_val_LR, y_val)
    score_list_LR.append([score,x])
```

In [63]:

```
pd.options.display.max_rows = 50
top_LR = pd.DataFrame(score_list_LR).sort_values(0, ascending = False).head(5)
top_LR
```

	0	1
47	0.826235	48
17	0.824532	18
18	0.824532	19
33	0.822828	34
31	0.822828	32

In [64]:

```
cols_2 = list(imp['columns'][0:top_LR[1].iloc[0]])
x_train = x_train[cols_2]
x_val = x_val[cols_2]
```

Final model and score with CV and gridsearch:

In [65]:

```
logreg = LogisticRegression(solver='newton-cg')
param = {'C':[1,10]}
LR = GridSearchCV(logreg,param,scoring='accuracy',refit=True,cv=10)
LR.fit(x_train,y_train)
score = LR.score(x_val, y_val)
print(LR.best_params_,score)
```

```
{'C': 10} 0.8262350936967632
```

In [ ]: