Project Report

Data Scientist Job Prediction Gajam Anurag

Problem Statement: The dataset contains information about people's jobs such as them demographic information etc. and the target variable is to predict whether the person is looking for a job change or not, the dataset is imbalanced.

Data Description:
There are 14,368 rows in the training set and 4790 rows in the testing set. The columns in the dataset are as follows

1. index:
   Unique ID for candidate
2. city:
   City code
3. city_ development _index:
   Development index of the city (scaled)
4. gender:
   Gender of candidate
5. relevent_experience:
   Relevant experience of candidate
6. enrolled_university:
   Type of University course enrolled if any
7. education_level:
   Education level of candidate
8. major_discipline :
   Education major discipline of candidate
9. experience:
   Candidate total experience in years
10. company_size:
    No of employees in current employer's company
11. company_type :
    Type of current employer
12. last*new*job:
    Difference in years between previous job and current job
13. training_hours:
    training hours completed
14. target:
    0 – Not looking for job change, 1 – Looking for a job change

Data Analysis and Visualization:

Training Data:

Categorical features: city, gender, relevant experience, enrolled university, education level, major discipline, experience, company size, company type, lastnewjob

Continuous features: city development index, training hours

The data set has missing values and below can see column wise NaN values

```
[7]  df.isna().sum()

index                      0
city                       0
city_development_index     0
gender                  3393
relevent_experience        0
enrolled_university      292
education_level          338
major_discipline        2089
experience                45
company_size            4430
company_type            4598
last_new_job             327
training_hours             0
target                     0
dtype: int64
```

Types of Data in Data Set:

Nominal Data: City, Gender, Enrolled_university, Major_discipline, Company_type, Target, Index

Ordinal Data: Education_level, Relevent_experience

Interval Data: City_development_index, Last_new_job

Ratio Data: Company_size, Experience, Training_hours

Basic information of continuous data in the DataFrame:

```
[49] df.describe()
```

	index	city_development_index	training_hours	target
count	14368.000000	14368.000000	14368.000000	14368.000000
mean	9634.231765	0.828252	65.396645	0.247982
std	5522.764568	0.123419	60.277583	0.431856
min	0.000000	0.448000	1.000000	0.000000
25%	4840.750000	0.738000	23.000000	0.000000
50%	9693.500000	0.899000	47.000000	0.000000
75%	14405.250000	0.920000	88.000000	0.000000
max	19157.000000	0.949000	336.000000	1.000000

Unique Values in the categorical features:

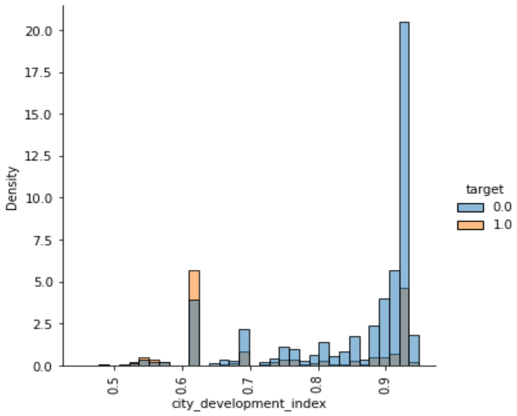
```
[6] cols = ['gender','relevent_experience','enrolled_university','education_level','major_discipline','company_type','experience','company_size','last_new_job']
    for col in cols:
      print(col,'-',df[col].unique(),end = '\n\n')
```

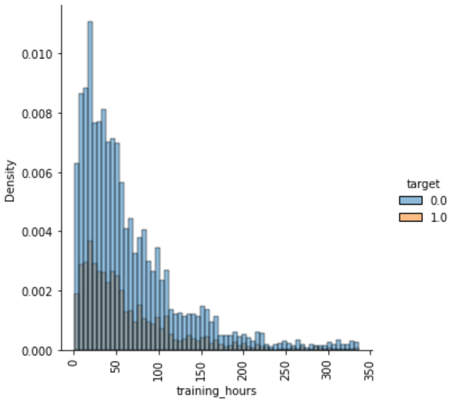
```
gender - ['Male' nan 'Female' 'Other']

relevent_experience - ['Has relevent experience' 'No relevent experience']

enrolled_university - ['no_enrollment' nan 'Full time course' 'Part time course']

education_level - ['Masters' 'High School' 'Graduate' 'Phd' nan 'Primary School']

major_discipline - ['STEM' nan 'No Major' 'Other' 'Arts' 'Humanities' 'Business Degree']

company_type - ['NGO' nan 'Pvt Ltd' 'Public Sector' 'Early Stage Startup'
 'Funded Startup' 'Other']

experience - ['4' '3' '9' '14' '1' '17' '>20' '12' '11' '13' '20' '7' '6' '15' '8' '19'
 '2' '10' '5' '16' '<1' '18' nan]

company_size - ['100-500' nan '50-99' '5000-9999' '10000+' '500-999' '<10' '10/49'
 '1000-4999']

last_new_job - ['1' '4' '2' '>4' 'never' '3' nan]
```

Data Visualization:

Relation between continuous features and target column:

Here I have visualized the continuous data (city_delelopment_index, training_hours) with respect to target variable with the help of seaborn plot called displot and below are the plots





Relation between categorical features and target column:

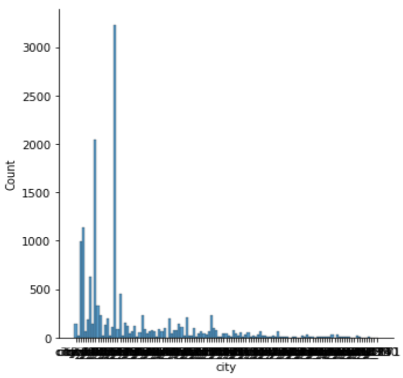
For categorical data I have used pd.crosstab with respect to target variables and below are few crosstab that are shown:

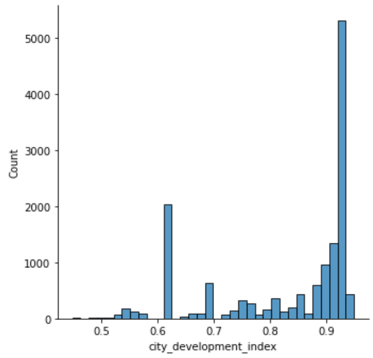
target	0.0	1.0
major_discipline		
Arts	155	40
Business Degree	162	64
Humanities	385	108
No Major	128	44
Other	201	67
STEM	8093	2832

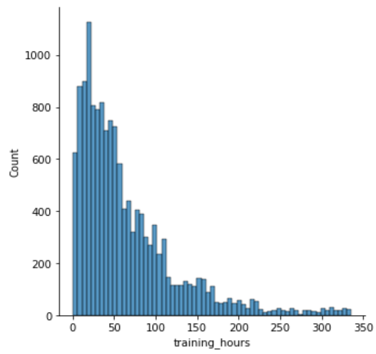
target	0.0	1.0
education_level		
Graduate	6271	2408
High School	1204	301
Masters	2606	705
Phd	261	49
Primary School	196	29

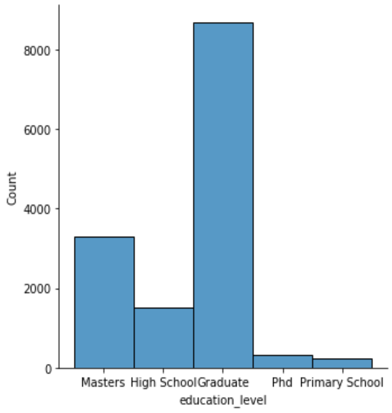
Distribution of features:

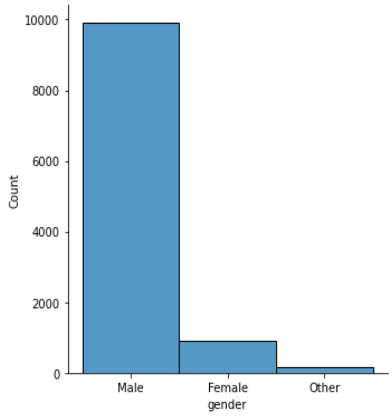
I have also visualized the distribution of individual column with the help of seaborn plot called displot. Few plots are shown below:

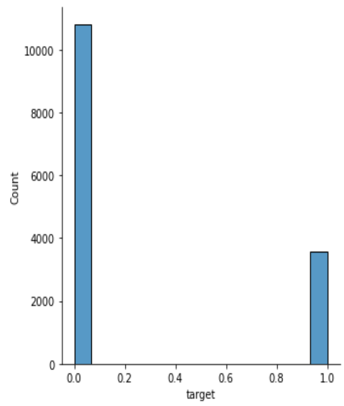












Finding correlation between continuous features and plotting heatmap:

```
[51] df.corr()
```

	index	city_development_index	training_hours	target
index	1.000000	0.001561	0.006070	0.005439
city_development_index	0.001561	1.000000	0.002239	-0.345985
training_hours	0.006070	0.002239	1.000000	-0.021143
target	0.005439	-0.345985	-0.021143	1.000000

```
52] import seaborn as sns
    sns.heatmap(df.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8a0c2d1110>
```

Data Pre-processing:

Dealing with missing values:

I have made few corrections to **experience, last_new_job, company_size** columns by replacing it with suitable number value and imputing 0 for missing values.

```
df4.experience.replace('>20',25,inplace=True)
df4.experience.replace('<1',0,inplace=True)
df4.experience.fillna(0,inplace=True)
df4.experience = df4.experience.astype('int')

df4.last_new_job.fillna(0,inplace=True)
df4.last_new_job.replace('>4',5,inplace=True)
df4.last_new_job.replace('never',0,inplace=True)
df4.last_new_job = df4.last_new_job.astype('int')

## company_size
df4.company_size.replace('10/49','10-49',inplace=True)
df4.company_size.replace('10-49',30,inplace=True)
df4.company_size.replace('<10',5,inplace=True)
df4.company_size.replace('50-99',75,inplace=True)
df4.company_size.replace('100-500',250,inplace=True)
df4.company_size.replace('500-999',750,inplace=True)
df4.company_size.replace('1000-4999',3000,inplace=True)
df4.company_size.replace('5000-9999',7000,inplace=True)
df4.company_size.replace('10000+',10000,inplace=True)
df4.company_size.fillna(0,inplace=True)
df4.company_size = df4.company_size.astype('int')
```

For remaining all other categorical features, I have imputed **Unknown** value for missing values:

```
df4.fillna('Unknown',inplace=True)
```

For **Education_Level** and **relevant_experience** column I have replaced values with number based on ranking.

```
cat = {"Phd":5, "Masters":4,"Graduate":3,"High School":2,"Primary School":1,"Unknown":0}
df4.education_level.replace(cat,inplace=True)
df4.education_level = df4.education_level.astype('int')

re = {"Has relevent experience":1,"No relevent experience":0}
df4.relevent_experience.replace(re,inplace=True)
df4.relevent_experience = df4.relevent_experience.astype("int")
```

Performed t-test among the continuous features:

```
[23] ## Performing ttest on continuous features
     data1 = df[df.target == 1]
     data0 = df[df.target == 0]

     print(ttest_ind(data1['training_hours'],data1['city_development_index']))

     print(ttest_ind(data0['training_hours'],data0['city_development_index']))
```

```
Ttest_indResult(statistic=65.12128727167367, pvalue=0.0)
Ttest_indResult(statistic=110.80067187547533, pvalue=0.0)
```

Performed chi-square test among the categorical features with respect to target.

Below are few results of the chi square test:

```
for i in cols:
  crosstable = pd.crosstab(df[i],df['target'])
  chival, pval, d, exp = chi2_contingency(crosstable)
  print(crosstable)
  print(chival, pval, d, exp,'\n')
```

```
target   0.0   1.0
gender
Female   684   230
Male    7644  2270
Other    107    40
3.8041120924187073 0.14926141515114838 2 [[ 702.46833713  211.53166287]
 [7619.55261959 2294.44738041]
 [ 112.97904328   34.02095672]]
```

```
target                     0.0   1.0
relevent_experience
Has relevent experience   8169  2216
No relevent experience    2636  1347
239.77850199755187 4.395437074135855e-54 1 [[7809.7108157 2575.2891843]
 [2995.2891843  987.7108157]]
```

```
target                 0.0   1.0
enrolled_university
Full time course      1739  1071
Part time course       670   230
no_enrollment         8195  2171
351.16473842350234 5.566034986241858e-77 2 [[2116.88263711  693.11736289]
 [ 678.00511509  221.99488491]
 [7809.1122478  2556.8877522 ]]
```

Testing Null Hypothesis with Chi square Test:

```
] res
```

```
{'city': 'accepted',
 'company_size': 'accepted',
 'company_type': 'accepted',
 'education_level': 'accepted',
 'enrolled_university': 'accepted',
 'experience': 'accepted',
 'gender': 'accepted',
 'last_new_job': 'accepted',
 'major_discipline': 'accepted',
 'relevent_experience': 'accepted'}
```

In the above chi square test(question12) between categorial variables and target varaible, with the help of chi2_contigency function, we get the pval and upon observing the critical value of the alpha value which is 0.05 with the degree of freedom in the chi square distribution table, every categorical feature is accepting the Null hypothesis which means there is no association with the feature and target varaible.

But from the question 5, with the distribution of data of every feature including categorical and continuous features, city column can be removed as it as same distribution of city_development_index and for the model, I have considered all the columns as it is clearly visible that there is some association with target features. And from the heatmap we can see there is some correlation between target and city_development_index.

Dealing with Imbalance Data:

I have tried balancing the data with Smote, RandomUnderSampler and RandomOverSampler and out of those 3 RandomUnderSampler outperformed with better accuracy but minor difference in between them.

```
df4.drop('city',axis=1,inplace=True)

### deal with inbalance data

from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler, NearMiss
smt = SMOTE(random_state=47)
oversample = RandomOverSampler()
undersample = RandomUnderSampler()
nmiss = NearMiss(version=2, n_neighbors=5)
X1 = df4.drop('target',axis=1)
Y1 = df4['target']
print(X1.shape,Y1.shape)
X1,Y1=undersample.fit_resample(X1,Y1)
print(X1.shape,Y1.shape)
```

Training the Model:

Trained the data with 5 different classification algorithms as mentioned below.

1. logisticRegression

```
classification_report:
[ ]                precision    recall  f1-score   support

         0.0       0.72      0.72      0.72      8652
         1.0       0.72      0.72      0.72      8636

    accuracy                           0.72     17288
   macro avg       0.72      0.72      0.72     17288
weighted avg       0.72      0.72      0.72     17288

confusion_matrix:
 [[6258 2394]
 [2387 6249]]

f1_score:  0.7233057468603507

classification_report:
              precision    recall  f1-score   support

         0.0       0.72      0.73      0.72      2118
         1.0       0.74      0.72      0.73      2204

    accuracy                           0.73      4322
   macro avg       0.73      0.73      0.73      4322
weighted avg       0.73      0.73      0.73      4322

confusion_matrix:
 [[1548  570]
 [ 612 1592]]

f1_score:  0.7292716445258818
```

2. SVM

```
              precision    recall  f1-score   support

         0.0       0.70      0.77      0.74      7876
         1.0       0.79      0.73      0.76      9412

    accuracy                           0.75     17288
   macro avg       0.75      0.75      0.75     17288
weighted avg       0.75      0.75      0.75     17288

0.759847099883663


              precision    recall  f1-score   support

         0.0       0.68      0.76      0.72      1938
         1.0       0.79      0.71      0.75      2384

    accuracy                           0.74      4322
   macro avg       0.74      0.74      0.73      4322
weighted avg       0.74      0.74      0.74      4322

0.7485714285714286
```

3. DecisionTreeclassifier:

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      8632
         1.0       1.00      1.00      1.00      8656

    accuracy                           1.00     17288
   macro avg       1.00      1.00      1.00     17288
weighted avg       1.00      1.00      1.00     17288

0.9993066789923735
```

```
              precision    recall  f1-score   support

         0.0       0.78      0.94      0.85      1792
         1.0       0.95      0.81      0.88      2530

    accuracy                           0.86      4322
   macro avg       0.86      0.88      0.86      4322
weighted avg       0.88      0.86      0.87      4322

0.875080076873799
```

4. RandomForestClassifier:

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      8625
         1.0       1.00      1.00      1.00      8663

    accuracy                           1.00     17288
   macro avg       1.00      1.00      1.00     17288
weighted avg       1.00      1.00      1.00     17288

0.9992492926026448
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.94      0.91      2029
         1.0       0.94      0.88      0.91      2293

    accuracy                           0.91      4322
   macro avg       0.91      0.91      0.91      4322
weighted avg       0.91      0.91      0.91      4322

0.9122412241224124
```

5. LinearRegression

```
              precision    recall  f1-score   support

           0       0.73      0.72      0.72      8877
           1       0.71      0.72      0.71      8411

    accuracy                           0.72     17288
   macro avg       0.72      0.72      0.72     17288
weighted avg       0.72      0.72      0.72     17288

0.714923619271445З

              precision    recall  f1-score   support

           0       0.73      0.71      0.72      2209
           1       0.71      0.73      0.72      2113

    accuracy                           0.72      4322
   macro avg       0.72      0.72      0.72      4322
weighted avg       0.72      0.72      0.72      4322

0.718960315618473
```

Among those algorithms, **RandomForest** performed better with high accuracy, followed by DecisionTree, SVM, logistic and Linear Regression. The Submitted Kaggle submission file is from trained RandomForest Algorithm.

Training the Model without using sklearn:

Written a code for **logistic regression** by using sigmoid function and used weights and bias to the data. Without sklearn, the model got accuracy of 62% on test data.

Splitting the data:

```
[176] ## Splitting the dataset into training and testing data
      df6 = pd.concat([X1,Y1],axis=1)

      # Shuffle your dataset
      shuffle_df = df6.sample(frac=1)

      # Define a size for your train set
      train_size = int(0.8 * len(df6))

      # Split your dataset
      train_set = shuffle_df[:train_size]
      test_set = shuffle_df[train_size:]

      ## splitting the data

      x_train, y_train = train_set.drop('target',axis=1), train_set['target']
      x_test, y_test = test_set.drop('target',axis=1), test_set['target']
```

Model Training:

```
def sigmoid(x):
  return 1/(1+np.exp(-x))

def fit(x, y,learning_rate,iterations,para):
    size = x.shape[0]
    weight = para["weight"]
    bias = para["bias"]
    for i in range(iterations):
      sigma = sigmoid(np.dot(x, weight) + bias)
      loss = -1/size * np.sum(y * np.log(sigma)) + (1 - y) * np.log(1-sigma)
      dW = 1/size * np.dot(x.T, (sigma - y))
      db = 1/size * np.sum(sigma - y)
      weight -= learning_rate * dW
      bias -= learning_rate * db

    para["weight"] = weight
    para["bias"] = bias
    print(para)
    return para

para = {}
para["weight"] = np.zeros(x_train.shape[1])
para["bias"] = 0

def train(x, y, learning_rate,iterations):
  params = fit(x, y, learning_rate, iterations ,para)
  return params

params = train(x_train, y_train, learning_rate = 0.02, iterations = 500)
```

Results:

```
[ ] y_pred = np.dot(x_test, params["weight"])+params["bias"]
    y_pred = sigmoid(y_pred)

    for i in range(len(y_pred)):
      if y_pred[i] >= 0.5:
        y_pred[i] = 1
      else:
        y_pred[i] = 0

    f1_score(y_pred,y_test)

    0.6219361228026737
```

Conclusion:

On performing the required data pre-processing operations on the given imbalanced data set, we see Random Forest out performed with better accuracy on test data with 91%. And with oversampling the imbalanced data Random Forest produced better results than under sampling by using inbuilt imblearn package and smote also gave better results but not as good as imblearn functions. Of the available models I conclude that Random Forest is the best model for given dataset to predict data scientist Job.