Project on HR Analytics using Machine Learning Models

- AIM: To create a Data Science project, where we'll be predicting whether the whether the candidate is going to be hired for Data Scientist role or not based on the data given which explains the whether the is having any relevant experience or not what's his/her education level and much more.
  This model will help HR's in predicting whether a candidate will work for the organization after undergoing training or not.

- Steps to be taken in the project is sub-divided into the following sections. These are:

  - Loading necessary libraries such as 'numpy', 'pandas', 'sklearn. model' etc.
  - Loading Dataset as a CSV file for training & testing the models.
  - Splitting the data set into independent & dependent sets.
  - Feature engineering using One-hot-encoder for cleaning the data set.
  - Checking if still any null values or any other data_types other than float and integers are present into the dataset or not.
  - Importing the train_test_split model from sklearn.model for splitting data into train & test sets.
  - Importing different kinds of classification models & then training those models with the help of .fit()
  - Predicting the trained models & then checking their accuracy of the model using confusion matrix & accuracy score.
  - Here we have also used the boosting technique's to improve the accuracy of the model.
  - Then finally, taken the boosted accuracy model as a best model to predict the hiring of candidate's.

- Steps of creating ML model:
- Step-1: Importing numpy as np & pandas as pd for loading and reading the data-set.

```
[1] import numpy as np
    import pandas as pd
    from matplotlib import pyplot as plt
    import seaborn as sns
```

- Step-2: Loading the csv-dataset in the variable name 'data_train'. Then viewing the data with 'head()' function.

```
data_train=pd.read_csv('/content/Train_Data.csv')
```

```
[3] data_train.head()
```

	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	major_discipline	experience	company_size	compar
0	city_21	0.624000	Male	No relevent experience	Full time course	Graduate	STEM	3	missing	
1	city_21	0.616795	Male	No relevent experience	Full time course	Graduate	STEM	<1	missing	
2	city_73	0.754000	Male	No relevent experience	Full time course	Graduate	STEM	<1	missing	
3	city_57	0.866000	Male	No relevent experience	no_enrollment	Phd	STEM	9	100-500	
4	city_21	0.624000	Male	Has relevent experience	no_enrollment	Masters	STEM	5	50-99	

```
[4] df=data_train
```

-Viewing dataset using '.head()' function and storing into simple name as 'df'.

➢ Step-3: Splitting the dataset into dependent & independent sets to train & test the models.

```
x=df.drop(['target'],axis=1)
y=df['target']
```

```
[6] x.head()
```

	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	major_discipline	experience	company_size	company_type	last_new_job	training_hours
0	city_21	0.624000	Male	No relevent experience	Full time course	Graduate	STEM	3	missing	Pvt Ltd	never	31.372145
1	city_21	0.616795	Male	No relevent experience	Full time course	Graduate	STEM	<1	missing	missing	1	63.988905
2	city_73	0.754000	Male	No relevent experience	Full time course	Graduate	STEM	<1	missing	missing	never	19.000000
3	city_57	0.866000	Male	No relevent experience	no_enrollment	Phd	STEM	9	100-500	Pvt Ltd	>4	53.000000
4	city_21	0.624000	Male	Has relevent experience	no_enrollment	Masters	STEM	5	50-99	Pvt Ltd	>4	108.000000

```
y.head()

0    1
1    1
2    1
3    0
4    1
Name: target, dtype: int64
```

➢ Step-3: Applying feature engineering using One-Hot-Encoding to clean the dataset by changing any categorical/abnormal values to numerical/binary values.

Feature engineering for one hot encoding

```
city=pd.get_dummies(x['city'])
gender=pd.get_dummies(x['gender'])
relevent_experience=pd.get_dummies(x['relevent_experience'])
enrolled_university=pd.get_dummies(x['enrolled_university'])
education_level=pd.get_dummies(x['education_level'])
major_discipline=pd.get_dummies(x['major_discipline'])
experience=pd.get_dummies(x['experience'])
company_size=pd.get_dummies(x['company_size'])
company_type=pd.get_dummies(x['company_type'])
last_new_job=pd.get_dummies(x['last_new_job'])
```

-First created dummies for every column which had string/abnormal values.

```
[9] #concanating this with dataframe
x=x.drop(['city','gender','relevent_experience','enrolled_university','education_level','major_discipline','experience','company_size','company_type','last_new_job'],ax
```

```
[10] x.head()
```

	city_development_index	training_hours
0	0.624000	31.372145
1	0.616795	63.988905
2	0.754000	19.000000
3	0.866000	53.000000
4	0.624000	108.000000

```
[11] x=pd.concat([x,city,gender,relevent_experience,enrolled_university,education_level,major_discipline,experience,company_size,company_type,last_new_job],axis=1)
```

-Concatenating the those columns by dropping from test dataset and then concatenate using 'pd.head()' function.

```
[12] x.head()
```

	city_development_index	training_hours	city_1	city_10	city_100	city_101	city_102	city_103	city_104	city_105	...	Public Sector	Pvt Ltd	missing	1	2	3	4	>4	missing
0	0.624000	31.372145	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0
1	0.616795	63.988905	0	0	0	0	0	0	0	0	...	0	0	1	1	0	0	0	0	0
2	0.754000	19.000000	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0
3	0.866000	53.000000	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	1	0
4	0.624000	108.000000	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	1	0

5 rows × 191 columns

-Showing the concatenated columns using 'x.head()' function.

- **Step-4**: Checking if any null values or data_types other than float & integers are present into the dataset or not.

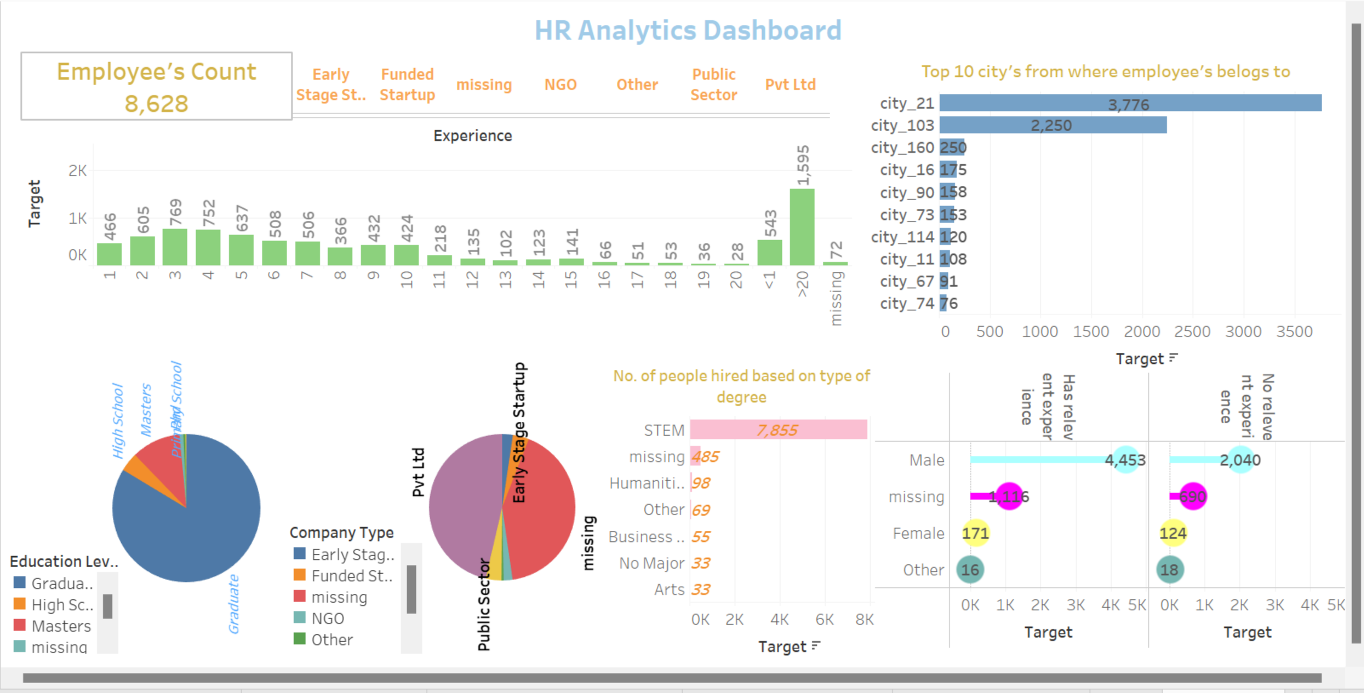
```
[15] x.dtypes

city_development_index    float64
training_hours            float64
city_1                      uint8
city_10                     uint8
city_100                    uint8
                           ...
3                           uint8
4                           uint8
>4                          uint8
missing                     uint8
never                       uint8
Length: 191, dtype: object
```

```
[16] x.isnull().sum()

city_development_index    0
training_hours            0
city_1                    0
city_10                   0
city_100                  0
                         ..
3                         0
4                         0
>4                        0
missing                   0
never                     0
Length: 191, dtype: int64
```

➢ Step-5: Visualizing the dataset to show the trend of hiring and based on multiple aspects.



-Also added a feature with which we can select individual company type to visualize how exactly they are hiring people, to know more about the visualization please click here

➢ Step-6: Importing train_test_split from sklearn.model_selection library for splitting the data into train and test sets.

Importing train_test_split model for training and testing the model

```
[17] from sklearn.model_selection import train_test_split
     x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

-Here 'test_size=0.2' means we are taking 20% data for testing the model & rest 80% for training the model.

➢ Step-7: Imported DecisionTreeClassifier then trained & predicted the x_test.

Importing DecisionTreeClassifier

```
[18] from sklearn.tree import DecisionTreeClassifier
     tree=DecisionTreeClassifier()

[19] tree.fit(x_train,y_train) #using fit() for training

     ▾ DecisionTreeClassifier
     DecisionTreeClassifier()

[20] predictions=tree.predict(x_test) #using .predict() to predict the o/p of the model

[21] predictions

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

➢ <u>Step-8</u>: Imported confusion_matrix and accuracy_score to check the accuracy of the model.

```
#importing confusion matrix & accuracy score for checking the accuracy of the model

from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_test,predictions)
acc=accuracy_score(y_test,predictions)

[23] print(cm)

[[1358  372]
 [ 362 1360]]

print(acc) #accuracy using decision tree // additional comment to be notes that before one hot encoding the acc was around 74 but now it's been raised to 78%

0.787369640787949
```

<u>-In the above model we can see that the accuracy obtained is 78% which can be better. So here I have also tried boosting techniques to boost the accuracy of the model.</u>

➢ <u>Imported AdaBoostClassifier & GradientBoostClassifier from 'sklearn.ensemble' library to Boost the accuracy of the model.</u>

Boosting the accuracy with Adaboost

```
[25] from sklearn.ensemble import AdaBoostClassifier
     ada_class=AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),n_estimators=20)

[26] ada_class.fit(x_train,y_train)

     ▸ AdaBoostClassifier
     ▸ estimator: DecisionTreeClassifier
       ▸ DecisionTreeClassifier

[27] ada_prediction=ada_class.predict(x_test)

[28] ada_prediction

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

```
[29] #importing confusion matrix & accuracy score again for checking the accuracy of the adaboost model

     from sklearn.metrics import confusion_matrix,accuracy_score
     CM=confusion_matrix(y_test,ada_prediction)
     ACC=accuracy_score(y_test,ada_prediction)

[30] print(CM)

     [[1435  295]
      [ 310 1412]]

[31] print(ACC)

     0.824739281575898
```

-In the above model the accuracy obtained using AdaBoost is 82% which is a bit better as compared to the model without Boosting technique.

Boosting the accuracy with Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
grad_class=GradientBoostingClassifier(learning_rate=0.1)
```

```
[33] grad_class.fit(x_train,y_train)

▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
[34] grad_prediction=grad_class.predict(x_test)
```

```
[35] grad_prediction

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

```
[36] #importing confusion matrix & accuracy score again for checking the accuracy of the GradientBoost model

from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_test,grad_prediction)
acc=accuracy_score(y_test,grad_prediction)
```

```
[37] print(cm)

[[1420  310]
 [ 260 1462]]
```

```
[38] print(acc)

0.8348783314020858
```

-Here we can see the accuracy obtained using GradientBoost is 83% which is again better from AdaBoosting.

- I have also used RandomForestClassifier model to check if we can obtain a much better accuracy for the model as compared to DecisionTreeClassifier.
- Imported RandomForestClassifier, then trained & trained & predicted the x_test.

Importing RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
```

```
[40] rf.fit(x_train,y_train)

▾ RandomForestClassifier
RandomForestClassifier()
```

```
[41] predictions=rf.predict(x_test)
```

```
[42] predictions

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

➤ Imported confusion matrix and accuracy score to check the accuracy of the model.

```
[43] from sklearn.metrics import confusion_matrix,accuracy_score
     cm=confusion_matrix(y_test,predictions)
     acc=accuracy_score(y_test,predictions)

[44] print(cm)

     [[1455  275]
      [ 287 1435]]

[45] print(acc)

     0.8371958285052143
```

-Here we have obtained accuracy of 83% which is much more better than DecisionTreeClassifier.

➤ Boosting the model using Ada & Gradient Boosting to check if we can get more accuracy for the model or not.

Boosting the accuracy with Adaboost

```
[46] from sklearn.ensemble import AdaBoostClassifier
     ada_rf=AdaBoostClassifier(RandomForestClassifier(max_depth=1),n_estimators=20)

[47] ada_rf.fit(x_train,y_train)

     ▸ AdaBoostClassifier
     ▸ estimator: RandomForestClassifier
       ▸ RandomForestClassifier

[48] ada_predictions=ada_rf.predict(x_test)

[49] ada_predictions

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

```
[50] #importing confusion matrix & accuracy score for checking the accuracy of the adaboost model

     from sklearn.metrics import confusion_matrix,accuracy_score
     cm=confusion_matrix(y_test,ada_predictions)
     acc=accuracy_score(y_test,ada_predictions)

[51] print(cm)

     [[1402  328]
      [ 295 1427]]

[52] print(acc) #here in randomforest when used adaboost it decreases the acc from 84% to 82%. So we'll try using gradientboost also

     0.8195249130938587
```

-Here using AdaBoosting into RandomForestClassifier we saw that we lost the accuracy by 2%. So will try using GradientBoosting.

Boosting the accuracy with Gradient Boosting

```
[53] from sklearn.ensemble import GradientBoostingClassifier
     grad_rf=GradientBoostingClassifier(learning_rate=0.1)

[54] grad_rf.fit(x_train,y_train)

     ▾ GradientBoostingClassifier
     GradientBoostingClassifier()

[55] grad_prediction=grad_rf.predict(x_test)

[56] grad_prediction

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

[57] #importing confusion matrix & accuracy score for checking the accuracy of the gradientboost model

     from sklearn.metrics import confusion_matrix,accuracy_score
     cm=confusion_matrix(y_test,grad_prediction)
     acc=accuracy_score(y_test,grad_prediction)

[58] print(cm)

     [[1419  311]
      [ 259 1463]]

[59] print(acc) #here after using Gradient Boosting the accuracy got increase very slightly. Which became same as the decision tree classifier's gradient boosting

     0.8348783314020858
```

-Here we have obtained accuracy of 83% which is completely same as DecisionTreeClassifier's Boosting score.

- Conclusion: From this project we have analysed and visualized what are trends of hiring based on candidates education types, company type, company size & much more & made a model for HR's to predict on the basis of data if they can hire the candidate as per companies hiring trend. So that it can save the time of HR's to optimize & screen out the uninterested candidates at the beginning itself.

THANK YOU