Configuration Manual

MSc Research Project
Data Analytics 2022-2023

Gayathri Malaichamy
Student ID:X21117683

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Gayathri Malaichamy
Student ID:	X21117683
Programme:	Data Analytics
Year:	2022-2023
Module:	MSc Research Project
Supervisor:	Anderson Simiscuka
Submission Due Date:	15/12/2022
Project Title:	Online Job Posting Authenticity Prediction using Machine and Deep Learning Techniques
Word Count:	643
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Gayathri Malaichamy
Date:	15th December 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Gayathri Malaichamy
X21117683

1 Introduction

Configuration Manual provides the details about system requirements, environment setup, tools and algorithms used for the Fake job prediction implementation. In this research, both Machine Learning and Deep Learning classifiers were used along with feature extraction techniques. To validate the model performance, evaluation metrics such as Accuracy, Time, F1-score, Precision and Recall were used. The process followed during the development phase as well as the final research findings are documented in the implementation section.

2 System Specification

The details of system specification are listed as follows.

- Operating System: Mac OS Ventura 13.0
- Chip: Apple M1
- Memory: 8GB
- Storage: Macintosh HD 494.38GB

Figure 1: System Specification

3 Software Requirements

This research work was implemented using Machine Learning and Deep Learning approaches and hence Python 3 was used for project implementation. For code implementation, Google Colaboratory was used as its faster and consists of plethora of libraries required for Deep Learning models.

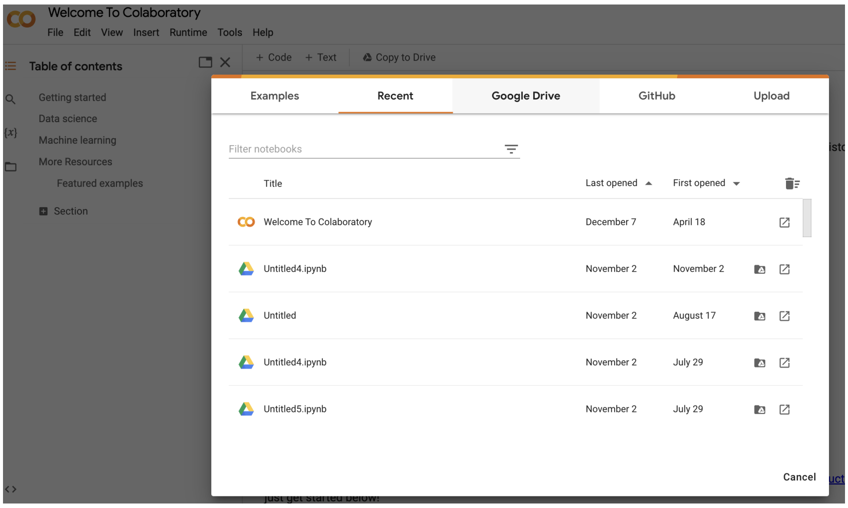

Figure 2: Google Colaboratory

4 Data Source

The fake job data was collected from the public repository called "Kaggle" [1]. This dataset consists of 17000 records and 18 columns. All these records are related to meta data about the job information.

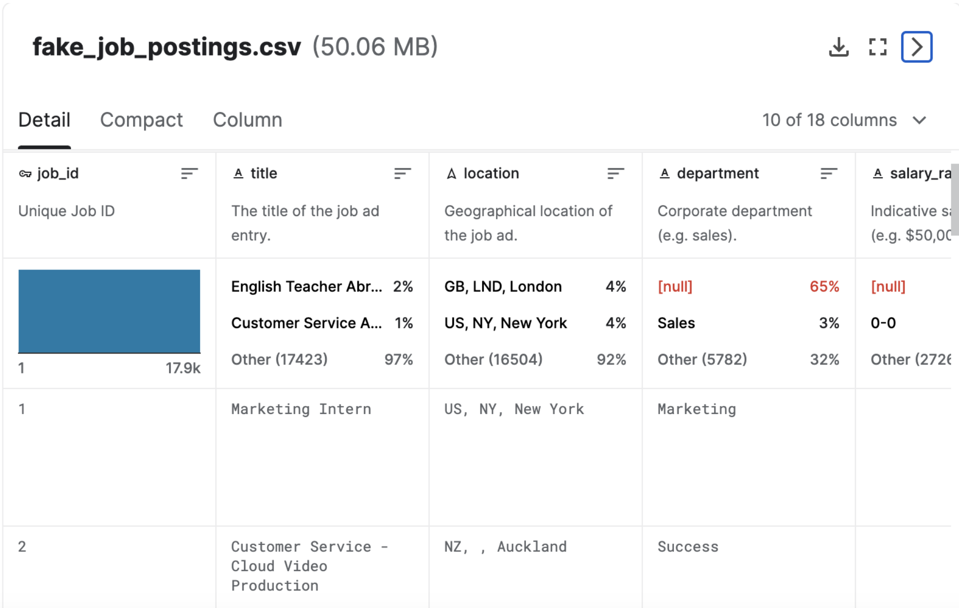

Figure 3: sourcedata

5 Data Load and Analysis

Firstly, important libraries were imported as shown in figure 4

[1] https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction

```
# Importing important Libraries:
import pandas as pd
import numpy as np
import nltk
import re
import seaborn as sns
import plotly.express as px
import spacy
import matplotlib.pyplot as plt
from spacy.lang.en.stop_words import STOP_WORDS
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from sklearn.model_selection import cross_val_score
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Figure 4: Important packages and Libraries

Fake job data was taken from Kaggle repository and kept in the Google Drive 5. From Google drive , the code was loaded in to Data input data frame by using pandas library as shown in figure 6

```
[ ] # Importing the fake jobdata file:
from google.colab import files
uploaded = files.upload()
```

Figure 5: Dataload from Google Drive

Data Load into Dataframe:

```
# Load the file into input dataframe:
data = pd.read_csv('/content/fake_job_postings.csv')
data.head()
```

	job_id	title	location	department	salary_range	company_profile	description	requirements	b
0	1	Marketing Intern	US, NY, New York	Marketing	NaN	We're Food52, and we've created a groundbreaki...	Food52, a fast-growing, James Beard Award-winn...	Experience with content management systems a m...	

Figure 6: dataload

The structure of the data is illustrated as shown in figure 6

```
print(data.shape,end='\n\n')
print(data.info())
```

```
(17880, 18)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17880 entries, 0 to 17879
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   job_id               17880 non-null  int64
 1   title                17880 non-null  object
 2   location             17534 non-null  object
 3   department           6333 non-null   object
 4   salary_range         2868 non-null   object
 5   company_profile      14572 non-null  object
 6   description          17879 non-null  object
 7   requirements         15185 non-null  object
 8   benefits             10670 non-null  object
 9   telecommuting        17880 non-null  int64
 10  has_company_logo     17880 non-null  int64
 11  has_questions        17880 non-null  int64
 12  employment_type      14409 non-null  object
 13  required_experience  10830 non-null  object
 14  required_education   9775 non-null   object
 15  industry             12977 non-null  object
 16  function             11425 non-null  object
 17  fraudulent           17880 non-null  int64
dtypes: int64(5), object(13)
memory usage: 2.5+ MB
None
```

Figure 7: structure of data

5.1 Exploratory Data Analysis

The Location column was splitted into city and country in order to plot the geographical distribution of fake job across the world. Also, various graphs were plotted to understand the fake job distribution.

```
[ ] # To split the location to extract the country in order to plot the geographical distribution:
    split_location=data["location"].apply(lambda x:str(x).strip().split(','))
    split_location=split_location.apply(pd.Series)
    data.shape
```

```
# Country values after mapping:
data['country']=data['country'].apply(lambda x:country_code_mapping[x] if x!='nan' else 'nan')
```

Figure 8: Extraction of country

```
fig = go.Figure(data=go.Choropleth(
    locations = list(percent_fraud_dict.keys()),
    z = list(percent_fraud_dict.values()),
    text =  list(percent_fraud_dict.keys()),
    colorscale = 'Reds',
    autocolorscale=False,
    marker_line_color='darkgray',
    marker_line_width=0.5,
    colorbar_title = 'Job ads percent',
))

fig.update_layout(
    title_text='Percentage of fraudulent job ads',
    geo=dict(
        showframe=False,
        showcoastlines=True,
        projection_type='equirectangular'
    ),

)

fig.show()
```

Figure 9: Total Fake jobs postings

```
# In terms of employment type, ads which specify "full-time", "contract" and "temporary" are less likely to be fake. For re

import seaborn as sb
data.dropna(axis= 0, how= 'any', inplace=True)
plt.figure(figsize=(10,8))
# sb.set_style("darkgrid")
sb.countplot(x='employment_type',data=data,palette='BuPu_r', hue = 'fraudulent')

## Interpretation based on results:
# From the plot, it is evident that the Full-time job opportunies have the highest number of fraudulent job advertisements.
```

Figure 10: Fake jobs postings for Employment_type

```
# Word Cloud_Mostly occured Keywords for fraudulent job postings:
#Separate fraud and actual jobs
fraudjobs_text = final[final.fraudulent==1].Text
STOPWORDS = spacy.lang.en.stop_words.STOP_WORDS
plt.figure(figsize = (16,14))                                            (parameter) stopwords: Any | None
wc = WordCloud(min_font_size = 3,  max_words = 3000 , width = 1600 , height = 800 ,background_color='white',stopwords = STOPWORDS).generate(str(" ".join(fra
plt.imshow(wc,interpolation = 'bilinear')
plt.axis("off")

# Interpretation based on results:
# Following Fake job word cloud shows that it consits of common general terms not job related requirements.
```

Figure 11: wordcloud for Fakjob postings

6 Data Preprocessing

This section explains various preprocessing steps that were implemented before model building. In this process, the basic operations such as unwanted column removal, HTML tags removal from the text were implemented. In addition, the NLP oprations such as stop words removal, special characters removal and tokenization were performed in order to get clean data for model building.

```
[ ] #Removing Unwanted Columns from the raw data

    data.drop(columns=['department','salary_range'],axis = 1,inplace=True)
```

```
[ ] data.fillna('',inplace=True)
```

Figure 12: IrrelevantColumn_Removal

```
[ ] # Removing HTML Tags:
    cols = ['company_profile' ,'description', 'requirements', 'benefits']

    for col in cols:
      for i in range(len(data[col])):
        TAG_RE = re.compile(r'<[^>]+>')
        data[col][i] = TAG_RE.sub('', data[col][i])
```

Figure 13: HTML_Tags_Removal

```
[ ] # Stopwords Removal from Text:
    all_stopwords = stopwords.words('english')
    print(all_stopwords)
    all_stopwords.remove('not')
    all_stopwords.remove('no')
    all_stopwords.remove('nor')

    ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll",
```

```
    # Special Characters removal from Text:
    import nltk
    nltk.download('omw-1.4')
    for i in range(0,final.shape[0]):
      sample = final['Text'][i]
      # Lemmatization process
      lemmatizer = WordNetLemmatizer()
      sample = re.sub("[^a-zA-Z']", ' ', sample)
      sample = re.sub(r"can\'t",'can not',sample)
      sample = re.sub(r"n\'t", " not", sample)
      sample = sample.lower()
      sample = sample.split()
      sample = [lemmatizer.lemmatize(word) for word in sample if not word in set(all_stopwords)]
      sample = ' '.join(sample)
      final['Text'][i] = sample
```

Figure 14: NLP Processing

Since the data is highly imbalanced, the RandomUnderSampling technique was applied in order to make the balanced data.

```
# Randomundersampler for data resampling:
columns = data.columns.tolist()
columns = [c for c in columns if c not in ["fraudulent"]]
target = "fraudulent"
state = np.random.RandomState(42)
X = data[columns]
Y = data["fraudulent"]

from imblearn.under_sampling import RandomUnderSampler

under_sampler = RandomUnderSampler()
X_rus, y_rus = under_sampler.fit_resample(X, Y)

df1 = pd.DataFrame(X_rus)
df2 = pd.DataFrame(y_rus)

result = pd.concat([df1, df2], axis=1, join='inner')
display(result)
data=result;
```

Figure 15: Random Under Sampling process

7 Model Building

This section provides a brief overview of model building process. For model building, both Machine Learning and Deep Learning classifiers were used. Before model build, the feature extraction techniques such as Unigram, Bigram, Trigram and TF-IDF were implemented.

```
# Unigram Model Building:
X = final.Text
y = final.fraudulent
from sklearn.feature_extraction.text import CountVectorizer
count_vectorize = CountVectorizer(ngram_range=(1,1))
X = count_vectorize.fit_transform(X).toarray()
```

Figure 16: Unigram model

```
[ ] # Train and Test data Split before model build:
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size = 0.20, random_state = 0)
```

Figure 17: Train and Test data Split

```
# Random Forest Classifier:
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score
import time

start_time = time.time()
forest = RandomForestClassifier()
forest.fit(X_train,y_train)
y_pred = forest.predict(X_test)
end_time = time.time()
eta = end_time - start_time
print('Random Forest Classifier')
# Evaluation metrics:
print('time taken:',eta)
print('accuracy :',accuracy_score(y_test,y_pred))
print('F1 score :',f1_score(y_test,y_pred))
print('Precision:',precision_score(y_test,y_pred))
print('Recall    :',recall_score(y_test,y_pred))
```

Figure 18: RandomForest Classifier

Random Forest classifier was retrained with hyper parameter tuning for optimized performance.

```
# Unigram_RandoForest with Hyperparameter Tuning:
start_time = time.time()
forest = RandomForestClassifier(criterion='entropy',n_estimators = 910, min_samples_split = 2, min_samples_leaf = 1, max_features = 'auto', max_
forest.fit(X_train,y_train)
y_pred = forest.predict(X_test)
end_time = time.time()
eta = end_time - start_time
print('Random Forest Classifier Uni-Gram with hyperparameter tuning')
# Evaluation metrics:
print('time taken:',eta)
print('accuracy :',accuracy_score(y_test,y_pred))
print('F1 score :',f1_score(y_test,y_pred))
print('Precision:',precision_score(y_test,y_pred))
print('Recall    :',recall_score(y_test,y_pred))
```

Figure 19: RandomForest with Hyperparameter Tuning

```
# Gaussian Naive Bayes :
from sklearn.naive_bayes import GaussianNB
start_time = time.time()
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_nb = gnb.predict(X_test)
end_time = time.time()
eta = end_time - start_time
print('Gaussian Naive Bayes')
# Evaluation metrics:
print('time taken:',eta)
print('accuracy :',accuracy_score(y_test,y_nb))
print('F1 score :',f1_score(y_test,y_nb))
print('Precision:',precision_score(y_test,y_nb))
print('Recall    :',recall_score(y_test,y_nb))
```

Figure 20: Naive Bayesn Classifier

```python
# Light GBM :
import lightgbm as lgb

start_time = time.time()
lgbm = lgb.LGBMClassifier()
lgbm.fit(X_train, y_train)
y_lgbm = lgbm.predict(X_test)
end_time = time.time()
eta = end_time - start_time
print('LightGBM Classifier')
# Evaluation metrics:
print('time taken:',eta)
print('accuracy :',accuracy_score(y_test,y_lgbm))
print('F1 score :',f1_score(y_test,y_lgbm))
print('Precision:',precision_score(y_test,y_lgbm))
print('Recall    :',recall_score(y_test,y_lgbm))
```

Figure 21: LightGBM Classifier

Light GBM classifier was retrained with hyper parameter tuning for optimized performance.

```python
#Unigram_LightGBM with Hyperparamenter tuning:
start_time = time.time()
lgbm = lgb.LGBMClassifier(subsample = 0.7000000000000001, random_state = 501, objective= 'binary', num_leaves= 30, min_split_gain = 0.4, min_dat
lgbm.fit(X_train, y_train)
y_lgbm = lgbm.predict(X_test)
end_time = time.time()
eta = end_time - start_time
# Evaluation metrics:
print('LightGBM Classifier')
print('time taken:',eta)
print('accuracy :',accuracy_score(y_test,y_lgbm))
print('F1 score :',f1_score(y_test,y_lgbm))
print('Precision:',precision_score(y_test,y_lgbm))
print('Recall    :',recall_score(y_test,y_lgbm))
```

Figure 22: LightGbm with Hyperparameter Tuning

```python
# XGBoost :
from xgboost import XGBClassifier
start_time = time.time()
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
y_xgb = xgb.predict(X_test)
end_time = time.time()
eta = end_time - start_time
# Evaluation metrics:
print('XGBoost Classifier')
print('time taken:',eta)
print('accuracy :',accuracy_score(y_test,y_xgb))
print('F1 score :',f1_score(y_test,y_xgb))
print('Precision:',precision_score(y_test,y_xgb))
print('Recall    :',recall_score(y_test,y_xgb))
```

Figure 23: XGBoost Classifier

In addition to Machine Learning models, Deep Learning models such as ANN and MLP classifiers were built for model comparison.

```python
# ANN :
import tensorflow
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
```

Figure 24: Important Libraries for Deep Learning Model

```
# ANN model building and prediction:
start_time = time.time()
model = Sequential()
model.add(Dense(units=100,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
model.fit(X_train,y_train,epochs=65,validation_data=(X_test, y_test), verbose=1)
end_time = time.time()
eta = end_time - start_time
print('time taken:',eta)
```

Figure 25: ANN Classifier

The MLP classifier was trained with two optimizers called "LBFGS" and "ADAM" in order to get optimized model.

```
#MLPClassifier with solver as lbfgs:

from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_auc_score

start_time = time.time()
mlp = MLPClassifier(solver='lbfgs',
                    activation = 'relu',
                   hidden_layer_sizes = (100,50,30),
                    max_iter = 1000)
mlp.fit(X_train, y_train)
mlp_pred = mlp.predict(X_test)
end_time = time.time()
eta = end_time - start_time
print('time taken:',eta)
# Evaluation metrics:
print('accuracy :',accuracy_score(y_test,mlp_pred))
print('roc_auc_score:', roc_auc_score(y_test, mlp_pred))
print('F1 score :',f1_score(y_test,mlp_pred))
print('Precision:',precision_score(y_test,mlp_pred))
print('Recall    :',recall_score(y_test,mlp_pred))
print(classification_report(y_test, mlp_pred))
```

Figure 26: MLP Classifier with LBFGS Optimizer

```
#MLPClassifier with solver as Adam:
start_time = time.time()
mlp = MLPClassifier(solver='adam',
                    activation = 'relu',
                   hidden_layer_sizes = (100,50,30),
                    max_iter = 1000)
mlp.fit(X_train, y_train)
mlp_pred = mlp.predict(X_test)
end_time = time.time()
eta = end_time - start_time
print('time taken:',eta)
# Evaluation metrics:
print('accuracy :',accuracy_score(y_test,mlp_pred))
print('roc_auc_score:', roc_auc_score(y_test, mlp_pred))
print('F1 score :',f1_score(y_test,mlp_pred))
print('Precision:',precision_score(y_test,mlp_pred))
print('Recall    :',recall_score(y_test,mlp_pred))
print(classification_report(y_test, mlp_pred))
```

Figure 27: MLP Classifier with ADAM Optimizer

Initially Unigram feature was implemented with all classifiers.Then, Bigram, Trigram and TF-IDF models were built with all classifiers in order to assess the model performance for different feature extraction techniques.

```
[ ] # Bi-gram mobel Building:

    X = final.Text
    y = final.fraudulent
    from sklearn.feature_extraction.text import CountVectorizer
    # Bigram range selection:
    count_vectorize = CountVectorizer(ngram_range=(2,2))
    X = count_vectorize.fit_transform(X).toarray()
```

Figure 28: Bigram model

```
[ ] # Tri_Gram Model building :
    X = final.Text
    y = final.fraudulent
    from sklearn.feature_extraction.text import CountVectorizer
    # Trigram range selection:
    count_vectorize = CountVectorizer(ngram_range=(3,3))
    X = count_vectorize.fit_transform(X).toarray()
```

Figure 29: Trigram_model

```
# TF-IDF model Building :
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(final.Text).toarray()
X
```

Figure 30: TF-IDF model

8 Model Evaluation and comparison

After model building, various evaluation metrics were used to evaluate the model performance. Evaluation metrics such as Accuracy, Time, F1_Score, Precision, Recall, Confusion Matrix and ROC_AUC curve were used.

```
print('time taken:',eta)
print('accuracy :',accuracy_score(y_test,mlp_pred))
print('roc_auc_score:', roc_auc_score(y_test, mlp_pred))
print('F1 score :',f1_score(y_test,mlp_pred))
print('Precision:',precision_score(y_test,mlp_pred))
print('Recall    :',recall_score(y_test,mlp_pred))
print(classification_report(y_test, mlp_pred))
```

```
MLP Classifier with lbfgs for TF_IDF model
time taken: 8.535385131835938e-05
accuracy : 0.930835734870317
roc_auc_score: 0.9312054849231178
F1 score : 0.9329608938547487
Precision: 0.943502824858757
Recall    : 0.9226519337016574
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       166
           1       0.94      0.92      0.93       181

    accuracy                           0.93       347
   macro avg       0.93      0.93      0.93       347
weighted avg       0.93      0.93      0.93       347
```

Figure 31: Evaluation_Metrics

```
# Confusion Matrix:
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
plt.figure(figsize=(4,3))
ConfMatrix = confusion_matrix(y_test,y_pred)
sns.heatmap(ConfMatrix,annot=True, cmap="Blues", fmt="d",
            xticklabels = ['Non_fraudulent', 'fraudulent'],
            yticklabels = ['Non_fraudulent', 'fraudulent'])
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion Matrix - Random Forest");
```

Figure 32: ConfusionMatrix

ROC_AUC Curve was used to compare both Machine Learning and Deep Learning models based on ROC_AUC score.

```
# Naive Bayes Classifier:
y_pred_proba_NB = gnb.predict_proba(X_test)[::,1]
fpr2, tpr2, _ = metrics.roc_curve(y_test,  y_pred_proba_NB)
auc2 = metrics.roc_auc_score(y_test, y_pred_proba_NB)

# LightGBM Classifier:
y_pred_proba_lgb = lgbm.predict_proba(X_test)[::,1]
fpr3, tpr3, _ = metrics.roc_curve(y_test,  y_pred_proba_lgb)
auc3 = metrics.roc_auc_score(y_test, y_pred_proba_lgb)

# XGBoost Classifier:
y_pred_proba_xgb = xgb.predict_proba(X_test)[::,1]
fpr4, tpr4, _ = metrics.roc_curve(y_test,  y_pred_proba_xgb)
auc4 = metrics.roc_auc_score(y_test, y_pred_proba_xgb)

# MLP Classifier:
y_pred_proba_mlp = mlp.predict_proba(X_test)[::,1]
fpr5, tpr5, _ = metrics.roc_curve(y_test,  y_pred_proba_mlp)
auc5 = metrics.roc_auc_score(y_test, y_pred_proba_mlp)

# ROC plot:
plt.figure(figsize=(10,7))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr1,tpr1,linestyle='--',label="Random Forest, auc="+str(round(auc1,2)))
plt.plot(fpr2,tpr2,linestyle='--',label="Naive Bayes Classifier, auc="+str(round(auc2,2)))
plt.plot(fpr3,tpr3,linestyle='--',label="LightGBM Classifier, auc="+str(round(auc3,2)))
plt.plot(fpr4,tpr4,linestyle='--',label="XGBoost Classifier, auc="+str(round(auc4,2)))
plt.plot(fpr5,tpr5,linestyle='--',label="MLP Classifier, auc="+str(round(auc5,2)))
plt.legend(loc=5, title='Models', facecolor='white')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC_AUC Curve', size=15)
plt.box(False)
plt.savefig('ImageName', format='png', dpi=200, transparent=True);
```

Figure 33: ROC_AUC_Curve Comparison