REAL / FAKE JOB POSTING PREDICTION

Information and Network Security

DINESH MARUPUDI VEERA – 999901286
VIVEK KIRAN KATARAM – 999902375
UDITHA REDDYKATTEGUMMULA - 999902201

1. Prior Work Evaluation:

The project involves predicting whether a job posting is real or fake using machine learning algorithms. Milestone 1 involved basic pre-processing of the data, while Milestone 2 focused on exploratory data analysis, feature selection, and feature importance. Here's an evaluation of the work done so far:

Milestone 1:

The basic pre-processing steps carried out on the dataset were crucial for ensuring the quality and accuracy of the machine learning model. The project team has successfully cleaned the data by removing missing values and duplicates, as well as transforming the categorical variables into numerical features using encoding. However, it would have been useful to see more details on how to handle outliers, as this could have a significant impact on the model's performance. Additionally, a more detailed explanation of the feature engineering process was required to conclude that the direction of work is precise.

Milestone 2:

The exploratory data analysis was carried out in milestone 2. we successfully visualized the data and identified key patterns and trends that could impact the model's accuracy. However, it would have been helpful to see more details on selected the features for the model, as well as the rationale behind the feature selection process. Additionally, it would have been beneficial to see a comparison of the performance of the different feature selection techniques used.

In terms of feature importance, we successfully identified the top features that are most important in predicting whether a job posting is real or fake. However, it would have been helpful to see more details on how to calculate the feature importance and the classification criteria used to rank the features.

Overall, we made significant progress in Milestones 1 and 2. However, there is still room for improvement in terms of providing more detailed explanations of the preprocessing steps, feature engineering, and feature selection processes. The improvements and necessary machine learning methods such as classifiers and ensemble models were applied to the pre-processed data to calculate the accuracy score of different features.

2. Data Preprocessing

The dataset (https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction) contains information related to job postings in different sources on the internet. This collection has 17,880 entries and 18 features. Out of which 800 are fake Job entries and the rest are Real job entries. The preprocessing steps include importing the required libraries that are

necessary to perform analysis on the dataset. After reading the dataset and identifying the target feature, we follow the steps below.

2.1. **Checking for missing values:** In order to fix the missing values in the dataset, we used SimpleImputer function shown below which helped to impute the numerical columns so that all the missing values will be filled automatically.

2.2. **Checking for duplicates**: After verification for duplicate values, all the duplicate rows that were present in dataset were dropped.

2.3 **Encoding categorical variables:** This involves converting categorical data into numerical values so that machine learning algorithms can process them. For dataset of job postings, encoder allows the algorithm to process the **job titles, industry codes, employment type and required experience** in predicting whether a job posting is real or fake

2.4 Drop Columns with low variance: If a feature has low variance, it means that it has a constant or almost constant value, making it irrelevant for machine learning algorithms. This technique helps in reducing the dimensionality of the dataset, improving the accuracy and efficiency of machine learning models.

In our dataset, since there were only 17 features, all of them have high importance in assessing the classification scores so none of the columns were dropped.

2.5 Clean_dataset Function:

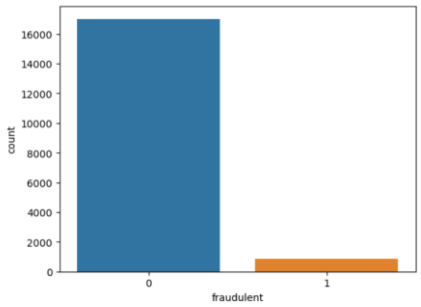
The given code defines a function 'clean_dataset' that takes a Pandas DataFrame 'df' as input and returns a cleaned version of the DataFrame by removing rows with null, NaN, and infinity values. The purpose of this code is to preprocess the given dataset to prepare it for training machine learning models, including cleaning the data, encoding categorical variables, and splitting the data into subsets for analysis.

3. Exploratory Data Analysis

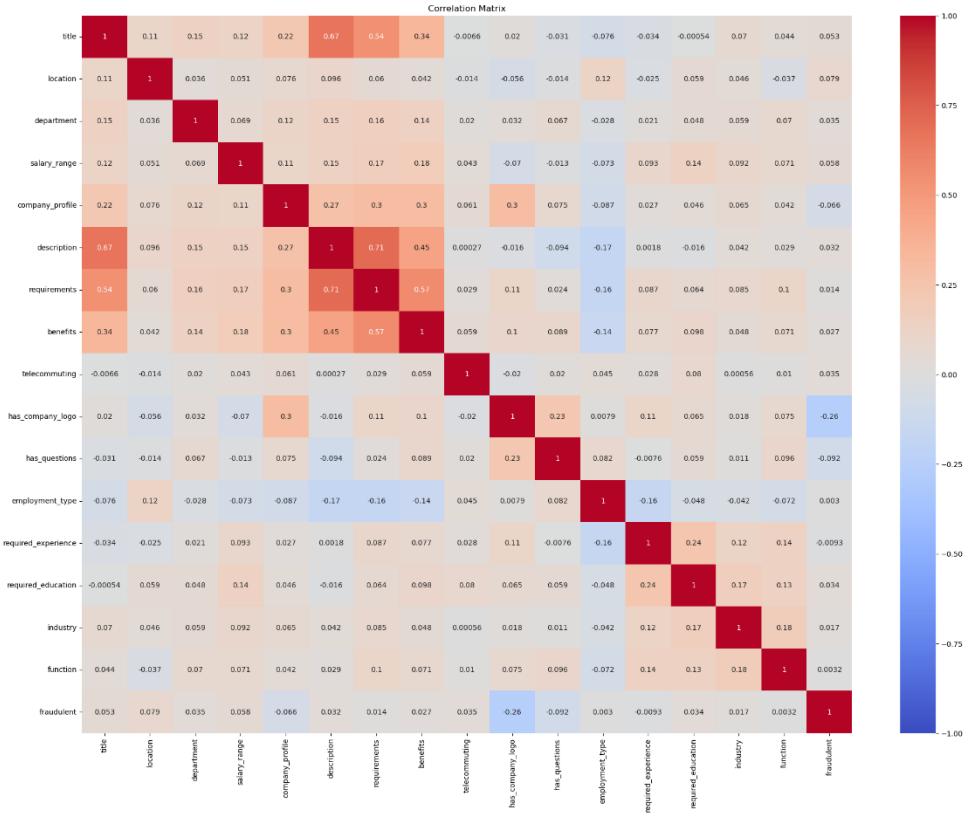
3.1 Splitting data into training and testing sets: In this project, we used 70% of the data for training and 30% of the data for testing. From code below we have defined X with all the features from the dataset except the target variable and the target variable is defined in y dataset.

```
# Separate the target variable and features
X = df.drop(targetVariable, axis=1)
y = df[targetVariable]

# Encode the target variable
lab_enc = preprocessing.LabelEncoder()
training_scores_encoded = lab_enc.fit_transform(y)
```

3.2. Class distribution: In this dataset, we have two classes - "**Real**" and "**Fake**". We can use a bar chart to visualize the class distribution. In plot below, 0 in X-tick represents the real jobs and 1 represents the fake entries in the dataset.

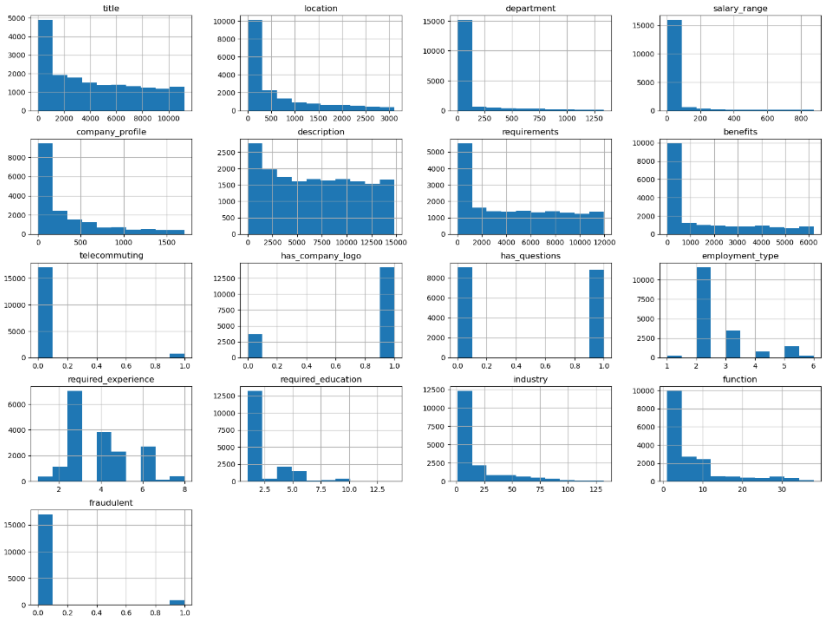


3.3. **Correlation between features:** A correlation matrix is a table that displays the pairwise correlations between all the variables in a dataset. It is a useful tool to identify the strength and direction of linear relationships between variables.

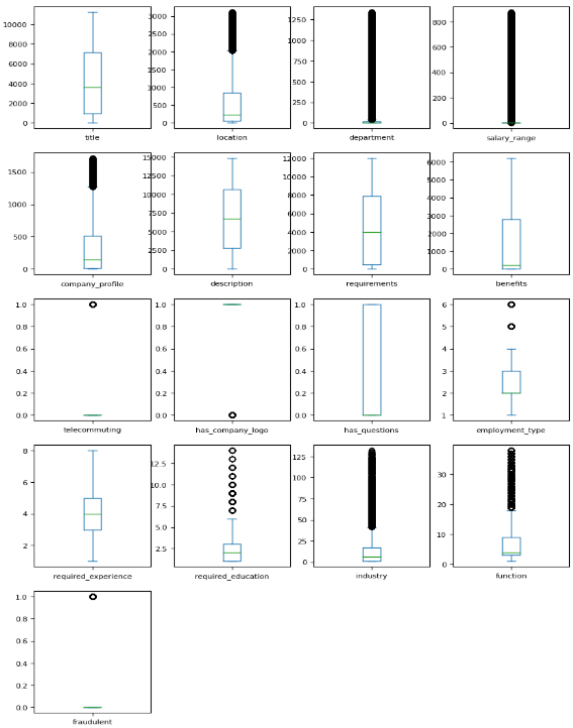


It helps in understanding the inter-dependencies among the variables, identifying the most important variables, and selecting features for the model. we analyzed the correlation between the features in the dataset and plotted a heatmap to visualize the correlation matrix.

3.4. **Skew of Univariate Distributions:** skewness is a measure of the asymmetry of a univariate probability distribution. A distribution is said to be skewed if it is not symmetric around its mean. Skewness can be positive, indicating a longer tail on the right side of the distribution, or negative, indicating a longer tail on the left side.



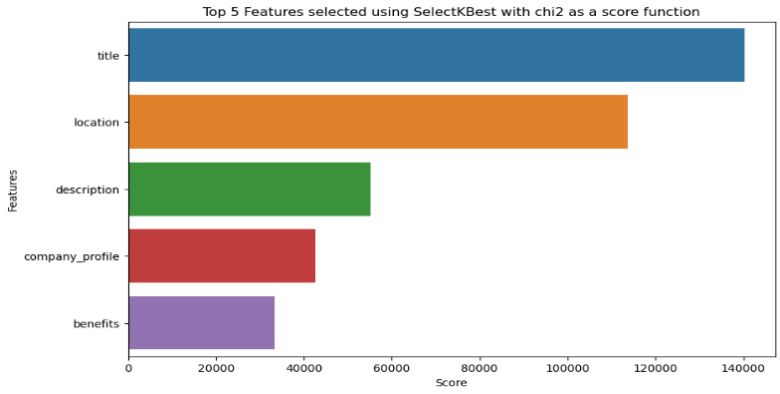
We also created a box plot for each column in the dataset to visualize the distribution of the data and identify any outliers or extreme values.



4. Feature Selection and Importance

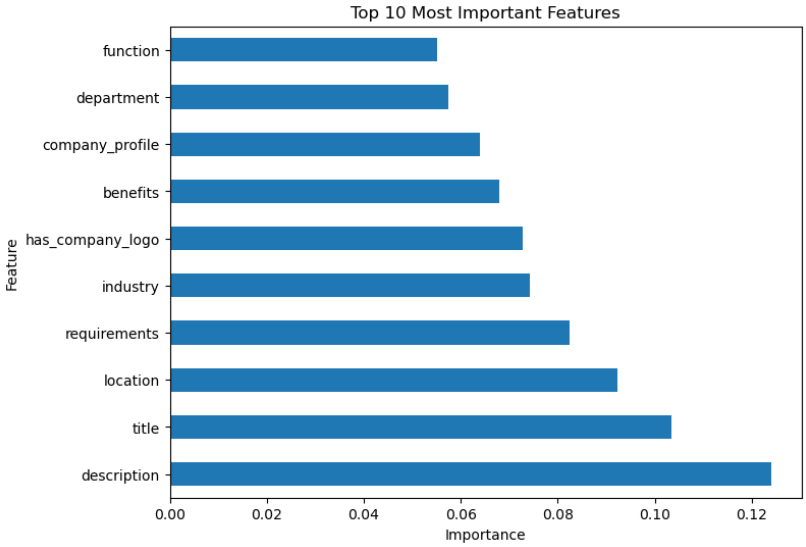
4.1. Feature selection

The template code aimed to perform feature selection using the SelectKBest method with **chi-squared (chi2) as the score function**. We applied SelectKBest to extract the top k features (where k = 'all' in this case). The result is the top 5 features selected by the SelectKBest method with chi2 as a score function and a bar chart was plottted.



4.2. Feature importance:

The code trains an **ExtraTreesClassifier model** to identify the **top 10 most important features** of a dataset. The purpose of this is to help identify which features have the greatest impact on the target variable, which can be useful in feature selection and model optimization. The output of the code provides insight into which features are most relevant for predicting the target variable.



5. Data Modeling

5.1. Model selection

In the given template code, **cross-validation technique** was used to evaluate the performance of a model and to ensure that the model is generalizing well on new, unseen data. K-fold cross-validation is a type of cross-validation technique that splits the dataset into k-folds and uses k-1 folds for training the model and the remaining one-fold for testing the model. This process is repeated k times with each fold used exactly once for testing the model.

The cross-validation parameters provided instantiates a KFold object with the following parameters:

- **n_splits:** The number of folds to be created, in this case, 10.
- **random_state:** The seed value used by the random number generator. This ensures that the same folds are generated each time the code is run.
- **shuffle**: Whether or not to shuffle the data before splitting it into folds.

The following were the list of classifiers for which the cross-validation technique is applied to perform model training and evaluation of each of the models.

- **Linear Regression**
- **Logistic Regression**
- **k-Nearest Neighbors**
- **Gaussian Naive Bayes**
- **Perceptron**
- **Linear SVC**
- **Stochastic Gradient Descent**
- **Decision Tree**
- **Random Forest**
- **Linear Discriminant Analysis**
- **Ada Boost - Ensemble**
- **Gradient Boosting**
- **ExtraTrees**
- **Extreme Gradient Boosting**

5.2. Model training:

The data is split into training and testing sets using the 'train_test_split' function from the 'model_selection' module of the scikit-learn library. This function randomly splits the data into two sets, one for training the model and another for testing the model. The testing set is usually a small percentage of the whole dataset, such as 20% in this case. The 'random_state' parameter is set to ensure that the same random split is used every time the code is executed.

```
# Separate the target variable and features
X = df.drop(targetVariable, axis=1)
y = df[targetVariable]

# Encode the target variable
lab_enc = preprocessing.LabelEncoder()
training_scores_encoded = lab_enc.fit_transform(y)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, training_scores_encoded, test_size=0.2, random_state=42)
```

6. Model Execution and Scores

6.1 Model Execution:

The code block below represents a sample model which is a part of a model execution script, that executes the logistic regression model using **cross-validation technique**. The input `X` and `training_scores_encoded` are feature set and target variable, respectively. The `LogisticRegression` function is used to initialize the logistic regression model with `lbfgs` solver and 1000 iterations. The `cross_val_score` function is then applied to the model, using the `X` and `training_scores_encoded` data with the defined `KFold` function, and the scoring metric is accuracy. In case of errors in model execution, the error gets displayed in the output.

```
if classifier_name == "LogR":
    clf_name = "Logistic Regression"
    try:
        # Logistic Regression
        kfold = KFold(n_splits=10, random_state=42, shuffle=True)
        start_time = time.perf_counter()
        logreg = LogisticRegression(solver='lbfgs', max_iter=1000)
        results_logreg = cross_val_score(logreg, X, training_scores_encoded, cv=kfold, scoring='accuracy')
        print('Estimated Accuracy of Logistic Regression: {:.2%}'.format(results_logreg.mean()))
        end_time = time.perf_counter()
        execution_time = end_time - start_time
        print(clf_name,'Executed in: {:.3f} seconds'.format(execution_time))
        execution_times[classifier_name] = execution_time
        accuracy = results_logreg.mean()
        results[clf_name] = accuracy

    except Exception as e:
        print("Error Logistic Regression")
        print (str(e))
```

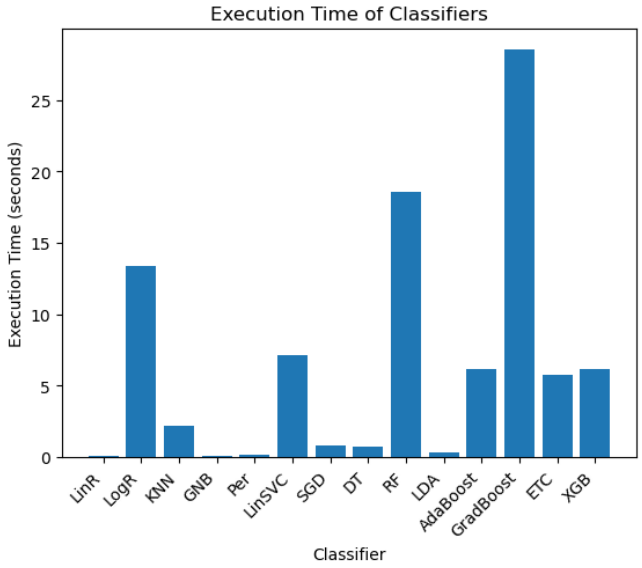
6.2. Model scores:

Upon model execution, the result will be calculated with **accuracy metrics** for each classifier. Accuracy is a commonly used evaluation metric for classification models. It is calculated as the ratio of the number of correct predictions to the total number of predictions. Image below shows the accuracy of each of the classifiers when cross validation technique is applied.

```
Classifier                      :: Accuracy
=====================================================
Linear Regression               ::  0.080
Logistic Regression             ::  0.952
k-Nearest Neighbors             ::  0.958
Gaussian Naive Bayes            ::  0.914
Perceptron                      ::  0.945
Linear SVC                      ::  0.860
Stochastic Gradient Descent     ::  0.937
Decision Tree                   ::  0.976
Random Forest                   ::  0.985
Linear Discriminant Analysis    ::  0.948
Ada Boost - Ensemble            ::  0.967
Gradient Boosting               ::  0.975
ExtraTrees                      ::  0.982
Extreme Gradient Boosting       ::  0.987
=====================================================
```

6.3 Model Execution Time:

Model execution time refers to the time it takes for a machine learning model to train on a dataset and make predictions on new data. Below image represents the execution time consumed by each of the classifiers for cross-validation technique. It is observed that Gradient Boost classifier consumed highest amount of time (~28seconds) while Linear Regression classifier takes the least ~ 0.28 seconds to execute.



7. Visualization of ROC Curve:

7.1. AUC Score:

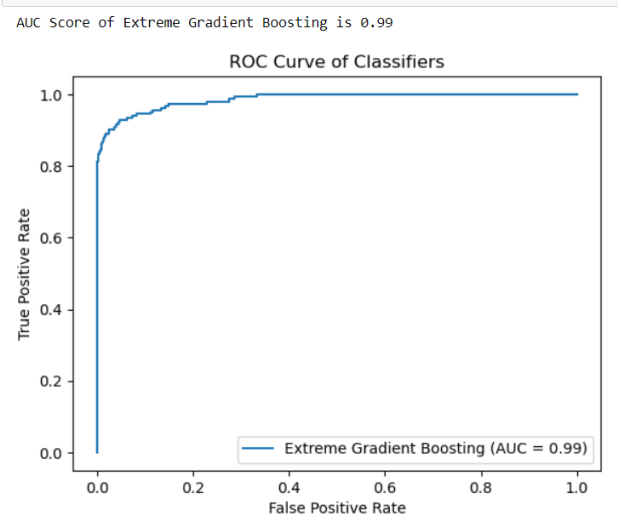
The AUC (Area Under the Curve) is the measure of the entire two-dimensional area underneath the ROC curve. It provides an aggregate measure of performance across all possible classification thresholds. The AUC-ROC curve ranges from 0.0 to 1.0, with higher values indicating better performance. An AUC-ROC score of 0.5 represents a random classifier, while a score of 1.0 represents a perfect classifier

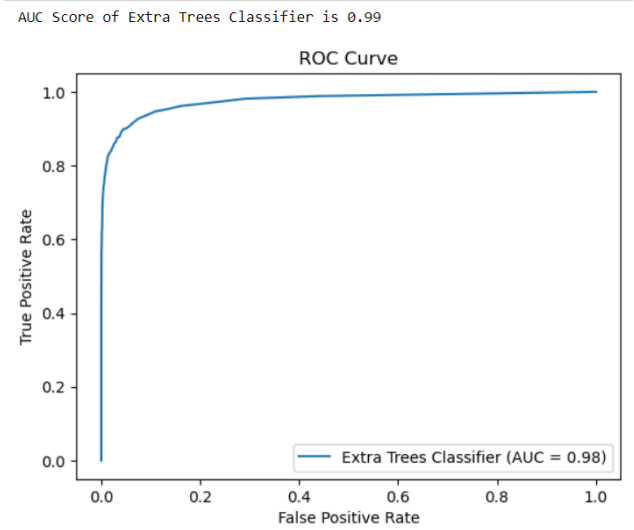
- **AUC Score for Extreme Gradient Boosting is 0.99 &**
- **AUC Score for Extra Trees Classifier is: 0.99**

7.2. ROC Curve:

The ROC curve is a graphical representation of the performance of a binary classifier system. It plots the true positive rate (TPR) against the false positive rate (FPR) at different classification thresholds. We used the roc_curve function from the sklearn library to generate the ROC curve.

Below plots show the **AUC Score and ROC Curve of top 2 classifiers that got high accuracy**.





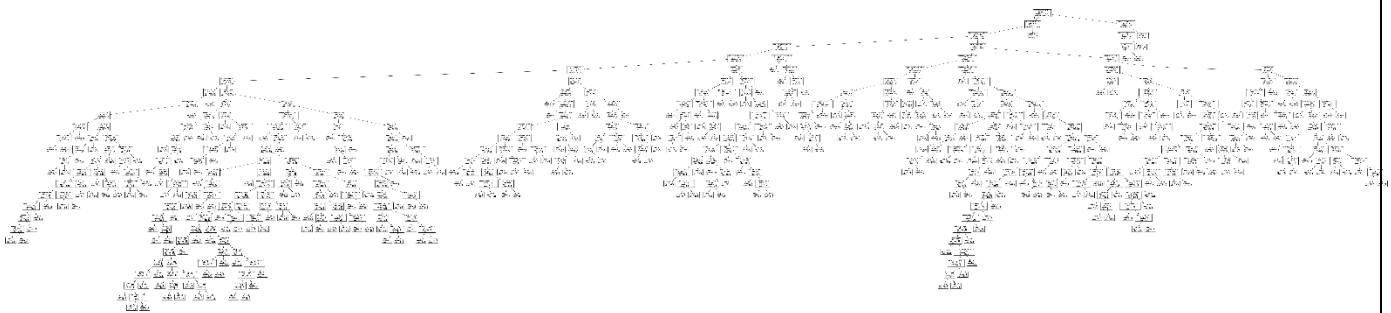
8. Ensemble Modelling

8.1. Ensemble modeling using Max Vote:

Ensemble modeling using Max Vote is a technique in which multiple models are trained and their predictions are combined to make the final decision.

The code defines a **Voting Classifier** model that combines the predictions of three different classifiers: Logistic Regression (model1), Random Forest (model2), and Decision Tree (model3). The "estimators" parameter specifies the list of models to be combined, along with a unique name for each model. **The "voting" parameter specifies the type of aggregation used to combine the predictions, which is set to 'hard' in this case**. With 'hard' voting, the final prediction is the most common prediction among the individual classifiers. The model can be trained and evaluated like any other scikit-learn classifier.

A decision tree dot file is a textual representation of a decision tree model in graphviz format. It is a plain text file that contains information about the nodes and edges of the tree. The dot file can be used to generate a visual representation of the tree using the graphviz software. It shows the hierarchy of the nodes, their decision criteria, and the predicted outcomes. The dot plot is a powerful tool for visualizing complex decision trees and can be used to debug and optimize the model.



Evaluation of Ensemble Model with voting classifier as “Max-Vote” for Decision Tree, Random Forest and Logistic regression is as below:

```
lenTestX= 5364  lenTesty= 5364
lenTrainX= 12516  lenTrainy= 12516
vote1= 0.9729679343773303
vote2= 0.9817300521998509
Actual: DecisionTree = 0.9729679343773303 , RandomForest = 0.9828486204325131 , LogisticReg = 0.9507829977628636
```

8.2 Ensemble Model – Average Vote:

Ensemble modeling is a technique in machine learning that combines multiple models to improve the accuracy and robustness of the predictions. This can be done by either combining different algorithms or training the same algorithm on different subsets of data. The most common types of ensemble models are bagging, boosting, and stacking.

Ensemble modeling can be especially useful in situations where individual models may have limitations, such as high bias or high variance, or where there may be a lack of data. However, it is important to note that ensemble models can be computationally expensive and require careful tuning of hyperparameters to achieve optimal performance.

Evaluation of Ensemble Model – Averaging with voting classifier as “Soft” for Decision Tree, Random Forest and Logistic regression is as below:

```
Model Ensemble Voting:Averaging
================================
Actual: Decision Tree = 0.97110365398956 , Random Forest = 0.9821029082774049 , LogisticReg = 0.9507829977628636
```

9. Application of NLP and Deep Learning Methods:

By using Natural Language Processing libraries, we have applied different operations on the pre-processed data with the main purpose of processing the raw text data into a format that can be used as input for machine learning models. Here are the characteristics and purpose of each step in the code:

9.1. **Prepare the data for training:**

This step involved loading and cleaning the raw data. The raw data is usually in the form of a dataset or a corpus of documents, which needs to be cleaned and preprocessed to remove any noise, such as HTML tags, punctuation, and stop words. The cleaned data is then stored in a panda DataFrame.

```python
# Define a function to clean the text data
def clean_text(text):
    if isinstance(text, str):
        text = text.lower()  # convert all the text into lowercase
        text = text.strip()  # remove starting and trailing whitespaces
        special_char_reg = '([a-zA-Z0-9]+)' + '[!"#$%&\'()*+,-./:;<=>?@\\^_`{|}~]' + '([a-zA-Z0-9]+)'
        text = re.sub(special_char_reg, ' ', text)
        text = re.sub(r'\s+', ' ', text) #remove all line formattings
        text = re.sub(r'\d+', '', text) #remove digits
        text = re.sub(r'\w*\d\w*', '', text)
        text = re.sub(r"\w+…|…", "", text)  # Remove ellipsis (and last word)
        text = re.sub(f"[{re.escape(string.punctuation)}]", "", text)
        text = ''.join(c for c in text if c not in string.punctuation)   #remove special symbols from job titles
        return text
    else:
        return ""
```

9.2. **Split the data into training and testing sets:**

This step involves dividing the data into two separate sets: training data and testing data. The training data is used to train the machine learning models, while the testing data is used to evaluate the performance of the models. This is done to ensure that the models generalize well to new, unseen data.

```
# Prepare the data for training
X = df["description_clean"]
y = df["fraudulent"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df["description_clean"], df["fraudulent"], test_size=0.2, random_state=42
```

9.3. **Tokenize the text data:**

Tokenization is the process of converting text into a sequence of tokens or words. In this step, the text data is tokenized using the Tokenizer class from the Keras API. The tokenizer maps each word in the text data to a unique integer, and creates a word index dictionary that can be used to convert the text data into sequences of integers. The num_words parameter specifies the maximum number of words to keep in the vocabulary, while the oov_token parameter represents any out-of-vocabulary words that were not seen during training.

```
# Tokenize the text data
vocab_size = 5000
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
```

9.4. **Convert the text data to sequences:**

This step involves converting the text data into sequences of integers using the word index dictionary created in the previous step. The resulting sequences are then used as input for machine learning models, which typically require numerical data. The sequences are also padded or truncated to ensure that they are of a fixed length, which is required by some models.

```
# Define the neural network architecture
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=32, input_length=max_length))
model.add(LSTM(32, return_sequences=True))
model.add(Dropout(0.5))
model.add(LSTM(32))
model.add(Dropout(0.5))
model.add(Dense(1, activation="sigmoid"))
```

Overall, this code is an essential part of the machine learning workflow for text classification tasks, and it helps to ensure that the text data is properly preprocessed and formatted for use in machine learning models.

Training the Model and Execution:

We have applied **Long Short-Term Memory (LSTM) neural network architecture** in Keras, for the task of binary text classification to detect fraudulent descriptions. In the Keras code, the LSTM model is defined using the Sequential model API, which allows the user to stack layers on top of each other in a linear

fashion. The input layer is followed by an Embedding layer, which maps each word in the input sequence to a dense vector representation.

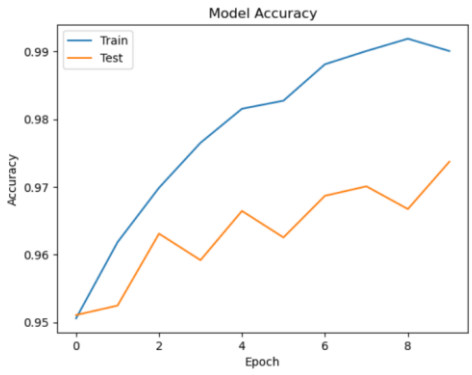
Finally, the output layer is a dense layer with a single neuron and a sigmoid activation function, which produces a binary output indicating whether the input is fraudulent or not.

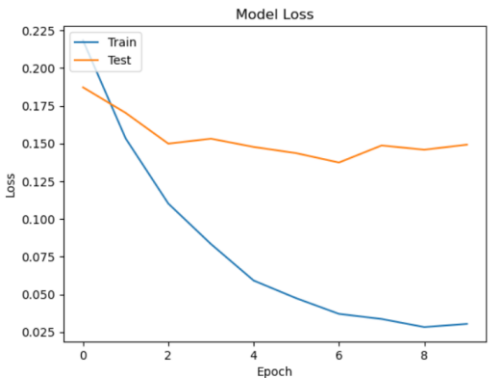
```
# Compile the model
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

# Train the model
history = model.fit(X_train_pad, y_train, epochs=10, batch_size=32, validation_data=(X_test_pad, y_test))
```

```
Epoch 1/10
447/447 [==============================] - 16s 33ms/step - loss: 0.2183 - accuracy: 0.9506 - val_loss: 0.1872 - val_accurac
y: 0.9511
Epoch 2/10
447/447 [==============================] - 19s 42ms/step - loss: 0.1534 - accuracy: 0.9618 - val_loss: 0.1703 - val_accurac
y: 0.9525
Epoch 3/10
447/447 [==============================] - 21s 47ms/step - loss: 0.1102 - accuracy: 0.9699 - val_loss: 0.1499 - val_accurac
y: 0.9631
Epoch 4/10
447/447 [==============================] - 23s 50ms/step - loss: 0.0833 - accuracy: 0.9765 - val_loss: 0.1532 - val_accurac
y: 0.9592
Epoch 5/10
447/447 [==============================] - 23s 51ms/step - loss: 0.0592 - accuracy: 0.9815 - val_loss: 0.1477 - val_accurac
y: 0.9664
Epoch 6/10
447/447 [==============================] - 23s 51ms/step - loss: 0.0474 - accuracy: 0.9827 - val_loss: 0.1436 - val_accurac
y: 0.9625
Epoch 7/10
447/447 [==============================] - 23s 50ms/step - loss: 0.0371 - accuracy: 0.9881 - val_loss: 0.1374 - val_accurac
y: 0.9687
Epoch 8/10
447/447 [==============================] - 27s 60ms/step - loss: 0.0337 - accuracy: 0.9901 - val_loss: 0.1487 - val_accurac
y: 0.9701
Epoch 9/10
447/447 [==============================] - 24s 54ms/step - loss: 0.0283 - accuracy: 0.9919 - val_loss: 0.1459 - val_accurac
y: 0.9667
Epoch 10/10
447/447 [==============================] - 22s 50ms/step - loss: 0.0304 - accuracy: 0.9901 - val_loss: 0.1492 - val_accurac
y: 0.9737
```

Visualize the LSTM Model Training Validations:

The plot in the left side represents the LSTM Model Accuracy on both test and train results, while the plot on the right represents the Model Loss for both Test and Train results.





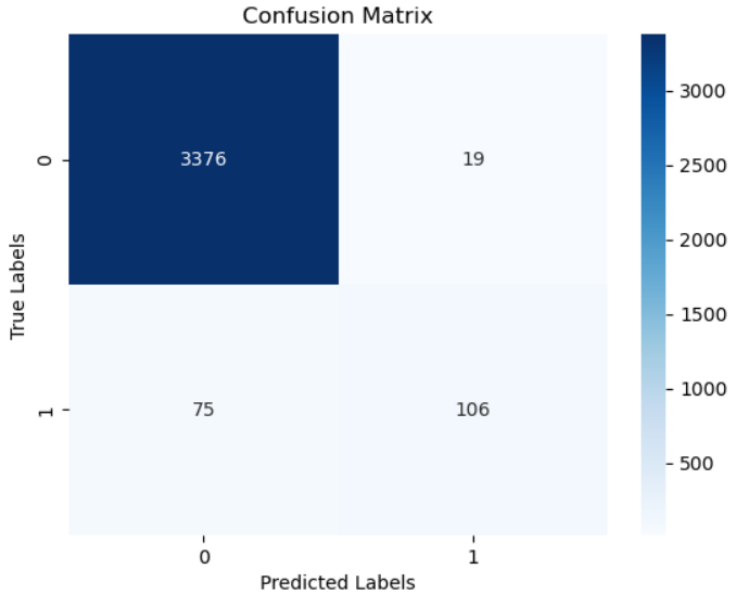
Model Evaluation:

The evaluation metrics provided in the output are precision, recall, and F1-score, which are common measures used to evaluate the performance of a binary classification model. Precision measures the proportion of true positive predictions out of all positive predictions, while recall measures the proportion of true positives out of all actual positive cases. The F1-score is the harmonic mean of precision

and recall and provides a balanced measure of the model's performance. The macro and weighted averages of these metrics are also provided, which give a summary of the overall performance of the model.

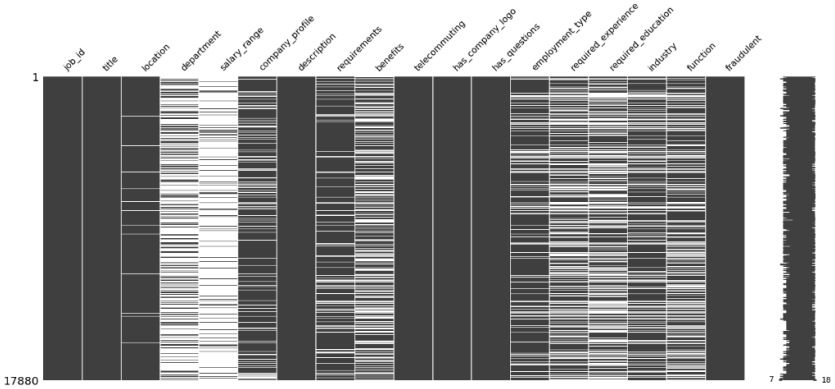
```
112/112 [==============================] - 2s 16ms/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3395
           1       0.85      0.59      0.69       181

    accuracy                           0.97      3576
   macro avg       0.91      0.79      0.84      3576
weighted avg       0.97      0.97      0.97      3576
```

Confusion Matrix: Below is the confusion matrix plotted for LSTM model using Keras.



10. Study on Kaggle Work

For a comparison study, we have selected Kaggle work: NLP(98%acc.) EDA with model using Spacy & Pipeline. This work is done by Shivam, Burnval. This code got 3rd most number of votes compared to 110 other Kaggle user works.

1. **Import Required libraries:** Author has imported necessary libraries to perform data analysis.
2. **Pre-Processing the data**:
   I. Author identified the missing values using **missingno** library and plotted the result.
   II. 

   III. Author has identified the target variable = '**fraudulent'**
   IV. In the next step, all the Nan Values has been filled in required columns and unnecessary columns has been deleted from the dataset.

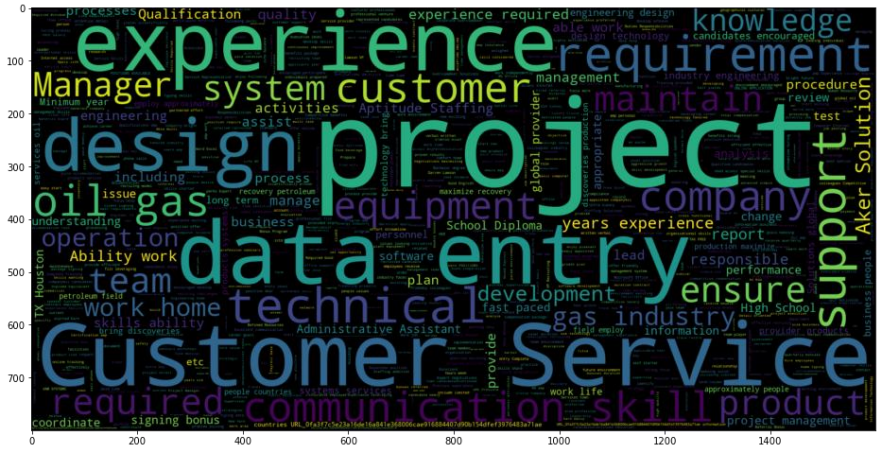
   a.
```
columns=['job_id', 'telecommuting', 'has_company_logo', 'has_questions', 'salary_range', 'employ
ment_type']
for col in columns:
    del data[col]

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

   V. As a part of cleaning the text data, Author had combined all the columns and stored in a new column with name: **data['text']** and deleted all other columns which means a new data model was created with only 2 features
      a. Target feature
      b. All features combined together (data[text])
3. **Feature Extraction:**
   - Author plotted all the words in real jobs and fake jobs using word cloud and stop words.

```
fraudjobs_text = data[data.fraudulent==1].text
actualjobs_text = data[data.fraudulent==0].text
```

```
STOPWORDS = spacy.lang.en.stop_words.STOP_WORDS
plt.figure(figsize = (16,14))
wc = WordCloud(min_font_size = 3,  max_words = 3000 , width = 1600 , height = 800 , stopwords =
STOPWORDS).generate(str(" ".join(fraudjobs_text)))
plt.imshow(wc,interpolation = 'bilinear')
```

Vii Example of one such plot is displayed below.



4. Cleaning the data model:

I. 2 functions were created such that it accepts a sentence as input and processes the sentence into tokens, performing lemmatization, lowercasing, and removing stop words.

```
# Create our list of punctuation marks
punctuations = string.punctuation

# Create our list of stopwords
nlp = spacy.load('en')
stop_words = spacy.lang.en.stop_words.STOP_WORDS

# Load English tokenizer, tagger, parser, NER and word vectors
parser = English()

# Creating our tokenizer function
def spacy_tokenizer(sentence):
    # Creating our token object, which is used to create documents with linguistic annotations.
    mytokens = parser(sentence)

    # Lemmatizing each token and converting each token into lowercase
    mytokens = [ word.lemma_.lower().strip() if word.lemma_ != "-PRON-" else word.lower_ for wor
d in mytokens ]

    # Removing stop words
    mytokens = [ word for word in mytokens if word not in stop_words and word not in punctuation
s ]

    # return preprocessed list of tokens
    return mytokens
```

```
# Custom transformer using spaCy
class predictors(TransformerMixin):
    def transform(self, X, **transform_params):
        # Cleaning Text
        return [clean_text(text) for text in X]

    def fit(self, X, y=None, **fit_params):
        return self

    def get_params(self, deep=True):
        return {}

# Basic function to clean the text
def clean_text(text):
    # Removing spaces and converting text into lowercase
    return text.strip().lower()
```

ii. After creating the functions, Author has performed cleaning the dataset.

5. **Training the model**:
   Author utilized 30% of data for testing and 70% of data for training.

```
# splitting our data in train and test
X_train, X_test, y_train, y_test = train_test_split(data.text, data.fraudulent, test_size=0.3)
```

6. **Creating Classifier Models:**
   i. Post creating the data Model, User has created Classifier model which accepts the input parameters of data model.

   ii. List of Classifier Models utilized were:
      a. **Logistic Regression:**
      b. **Random Forest Classifier**
      c. **Simple Vector Machine (SVM) Classifier**
      d. **XGBoost Classifier**

   iii. Screenshot below shows the implementation of classifier model by passing the data model using pipeline.

```
clf = LogisticRegression()

# Create pipeline using Bag of Words
pipe = Pipeline([("cleaner", predictors()),
                 ('vectorizer', bow_vector),
                 ('classifier', clf)])

# fitting our model.
pipe.fit(X_train,y_train)
```

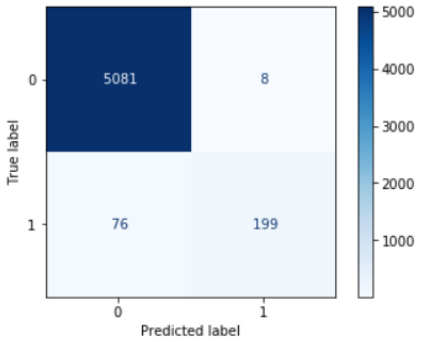
7. **Evaluation Metrics:**
   i. Author primarily used Accuracy and Recall as evaluation metrices for the classifier models. Screenshot below shows the Evaluation metrices.

```
# Predicting with a test dataset
predicted = pipe.predict(X_test)

# Model Accuracy
print("Logistic Regression Accuracy:", accuracy_score(y_test, predicted))
print("Logistic Regression Recall:", recall_score(y_test, predicted))
```

```
Logistic Regression Accuracy: 0.9843400447427293
Logistic Regression Recall: 0.7236363636363636
```

8. **Visualization and interpretation:** User only used confusion matrix to plot the model evaluation scores.

```
]: plot_confusion_matrix(pipe, X_test, y_test, cmap='Blues', values_format=' ')

]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fe939d11e90>
```

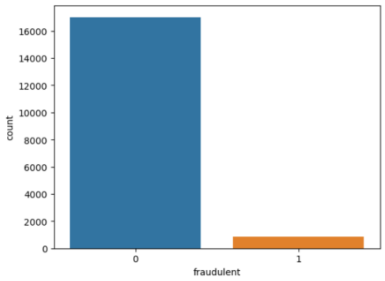


10.1 Comparative evaluation of Milestone-3 for Template Code:

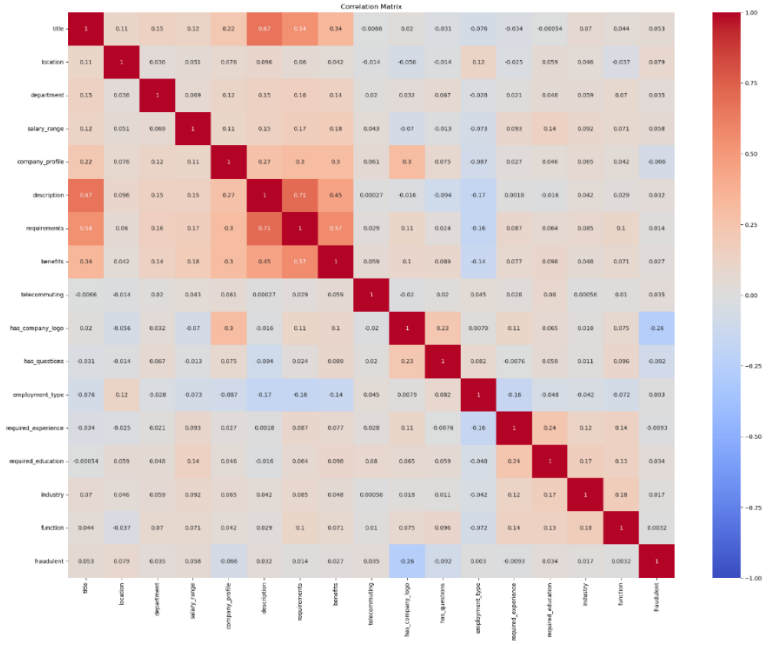
1. **Importing Required Libraries**: All the required libraries required to perform various data processing and analytical operations were imported.
2. **Pre-processing:**
   I. **Checking for missing values**: Using SimpleImputer function on the numerical columns filled the missing values.
   II. **Checking for duplicates**: All the duplicate rows that were present in dataset were dropped.
   III. **Encoding categorical variables**: This involves converting categorical data into numerical values.
   IV. **Drop Columns with low variance**: A new function was created to delete feature which have low variance to avoid false predictions for machine learning algorithms.
   V. **Clean_dataset Function**: This function takes the DataFrame 'df' as input and returns a cleaned version of the DataFrame by removing rows with null, NaN, and infinity values.
3. **Exploratory Data Analysis:**
   iv. **Splitting data into training and testing sets:** In this project, we used 70% of the data for training and 30% of the data for testing.

```
# Separate the target variable and features
X = df.drop(targetVariable, axis=1)
y = df[targetVariable]

# Encode the target variable
lab_enc = preprocessing.LabelEncoder()
training_scores_encoded = lab_enc.fit_transform(y)
```

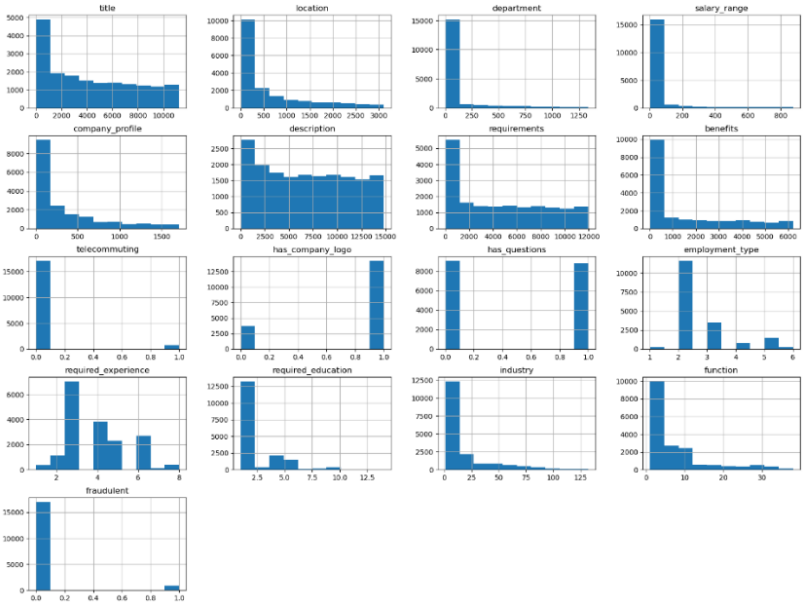
   v. **3.2. Class distribution:** In this dataset, we have two classes - "**Real**" and "**Fake**". We can use a bar chart to visualize the class distribution.



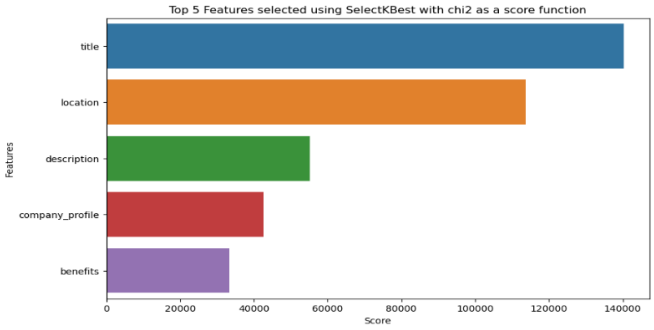
vi. 3.3. **Correlation between features:** A correlation matrix is a table that displays the pairwise correlations between all the variables in a dataset.



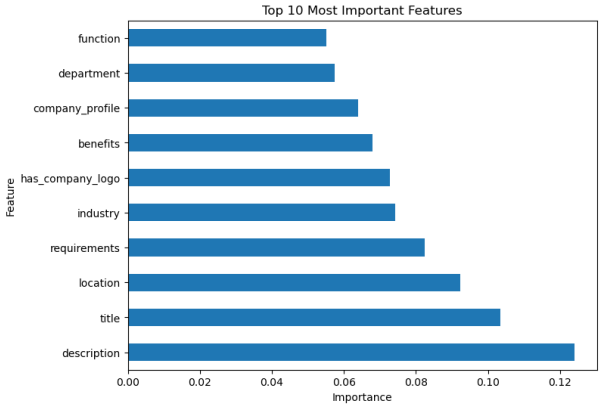
vii. **Skew of Univariate Distributions:** Skewness can be positive, indicating a longer tail on the right side of the distribution, or negative, indicating a longer tail on the left side.



4. **Feature selection:** The template code aimed to perform feature selection using the SelectKBest method with **chi-squared (chi2) as the score function**. The result is the top 5 features selected by the SelectKBest method with chi2 as a score function and a bar chart was plotted.



ii. **Feature importance:** The code trains an **ExtraTreesClassifier model** to identify the **top 10 most important features** of a dataset.

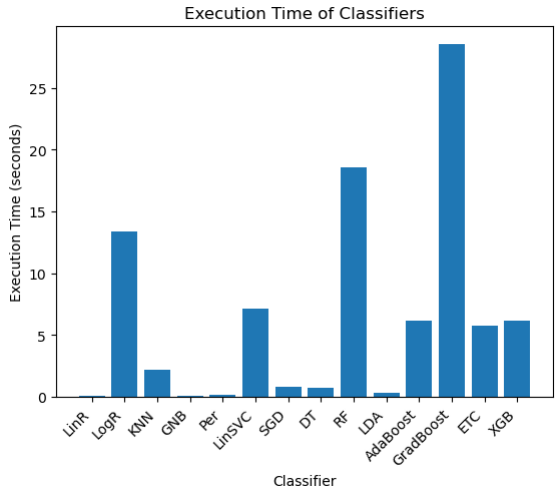


5. **Classifier Models:** The following were the list of classifiers for which the cross-validation technique is applied to perform model training and evaluation of each of the models.

- Linear Regression
- Logistic Regression
- k-Nearest Neighbors
- Gaussian Naive Bayes
- Perceptron
- Linear SVC
- Stochastic Gradient Descent
- Decision Tree
- Random Forest
- Linear Discriminant Analysis
- Ada Boost - Ensemble
- Gradient Boosting
- ExtraTrees
- Extreme Gradient Boosting

6. **Model Execution:** We used **cross-validation technique** which is a part of a model execution script, that executes all the classifier models. The model uses X, `training_scores_encoded` data with the defined `KFold` function as parameters.

7. **Model Execution Times:** Below image represents the execution time consumed by each of the classifiers for cross-validation technique.

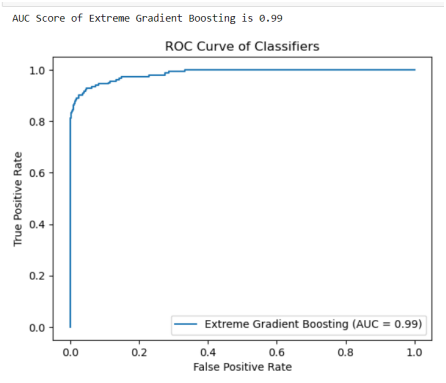


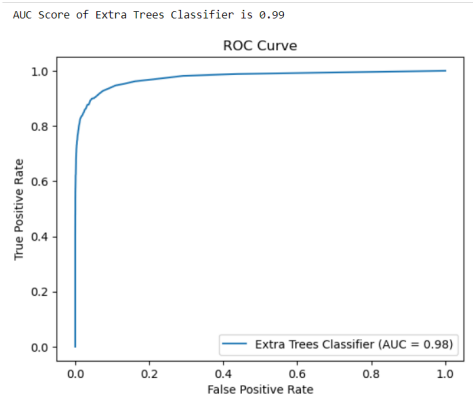
8. **Model Evaluation Metrics:**
   a. **Accuracy:** Accuracy of all the classifier models has been calculated as below.

Classifier	:: Accuracy
Linear Regression	:: 0.080
Logistic Regression	:: 0.952
k-Nearest Neighbors	:: 0.958
Gaussian Naive Bayes	:: 0.914
Perceptron	:: 0.945
Linear SVC	:: 0.860
Stochastic Gradient Descent	:: 0.937
Decision Tree	:: 0.976
Random Forest	:: 0.985
Linear Discriminant Analysis	:: 0.948
Ada Boost - Ensemble	:: 0.967
Gradient Boosting	:: 0.975
ExtraTrees	:: 0.982
Extreme Gradient Boosting	:: 0.987

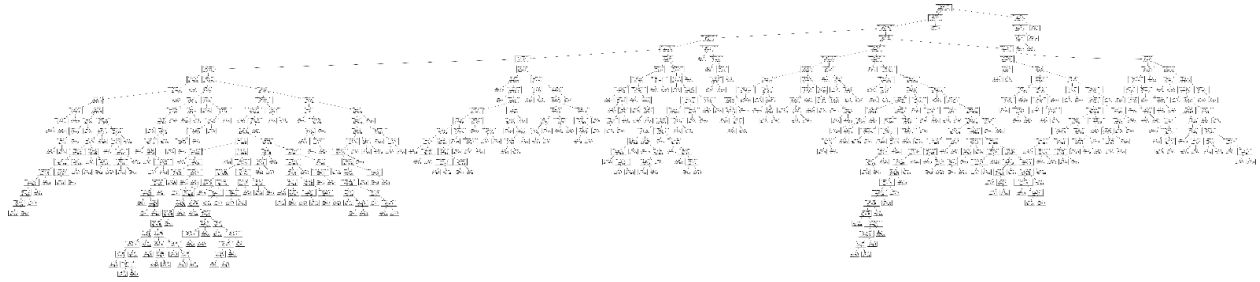
   b. **AUC Score:**
      - AUC Score for Extreme Gradient Boosting is 0.99 &
      - AUC Score for Extra Trees Classifier is: 0.99

c. **ROC Curve:** Below plots show the **ROC Curve of top 2 classifiers that got high accuracy**.





9. **Ensemble modeling using Max Vote:** It is a technique in which multiple models are trained and their predictions are combined to make the final decision.
    - Decision Tree is plotted using Ensemble model voting classifier on decision tree classifier.



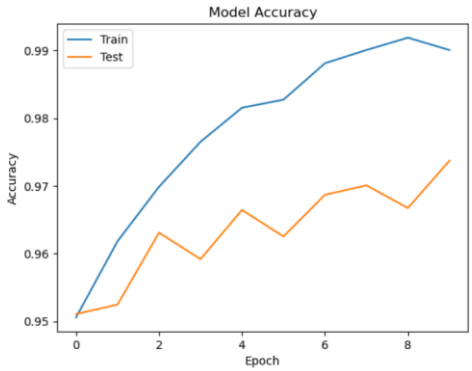
- Accuracy for Ensemble modelling max-vote is shown below.

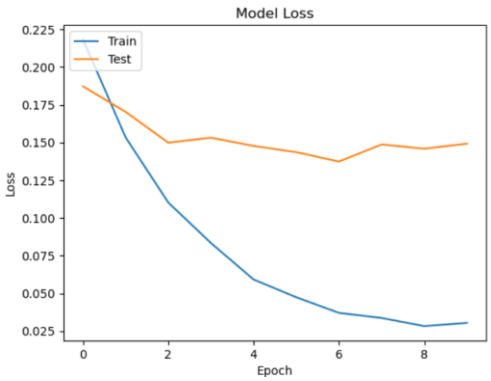
```
lenTestX= 5364  lenTesty= 5364
lenTrainX= 12516  lenTrainy= 12516
vote1= 0.9729679343773303
vote2= 0.9817300521998509
Actual: DecisionTree = 0.9729679343773303 , RandomForest = 0.9828486204325131 , LogisticReg = 0.9507829977628636
```

- Accuracy for Ensemble modelling Averaging is shown below.

```
Model Ensemble Voting:Averaging
================================
Actual: Decision Tree = 0.97110365398956 , Random Forest = 0.9821029082774049 , LogisticReg = 0.9507829977628636
```

10. **Application of Deep learning method: LSTM**
    - **LSTM Model Training Validations:** The plot in the left side represents the LSTM Model Accuracy on both test and train results, while the plot on the right represents the Model Loss for both Test and Train results.

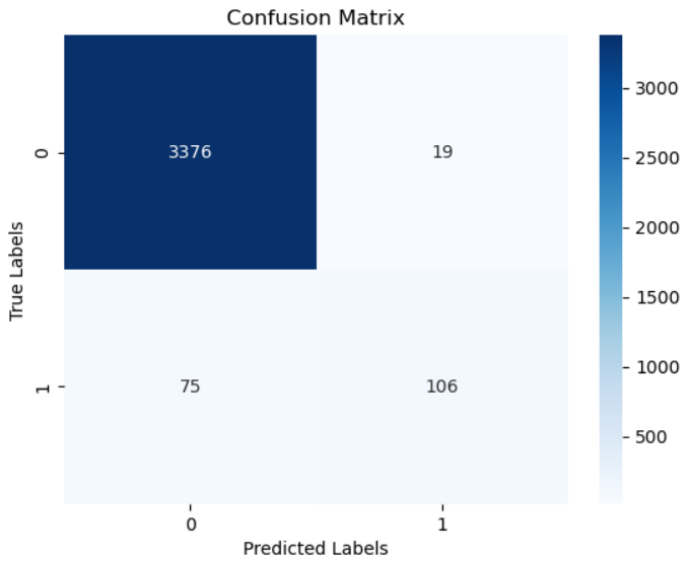




- **LSTM Model Evaluation:** The evaluation metrics provided in the output are precision, recall, and F1-score, which are common measures used to evaluate the performance of a binary classification model

```
112/112 [==============================] - 2s 16ms/step
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      3395
           1       0.85      0.59      0.69       181

    accuracy                           0.97      3576
   macro avg       0.91      0.79      0.84      3576
weighted avg       0.97      0.97      0.97      3576
```

- **Confusion Matrix for LSTM Model:**



11. <u>Conclusion of Comparison Study:</u>

S.No	Kaggle User's Efforts	Our Team's Efforts
1	Author has prepared Data Model with only target feature and all other features merged to a single feature	Strict Data Pre-processing using multiple functions
2	Feature Selection and Feature importance is not properly processed	Feature selection and Feature importance is achieved by chi squared as score function
3	Pre-processing of Numerical Columns is not clear	SimpleImputer function is used to impute all numerical columns
4	Very few classifiers models were used to evaluate the dataset	A wide range of classifier models were used to evaluate dataset

5	Author has created custom Classifier Model using Spacy and pipelines	We used standard Cross-Validation technique on classifier models
6	Achieved good accuracy in most classifiers	Achieved >95% accuracy score for most of the classifiers
7	Less interpretation of Model evaluation metrics	Wide range of model evaluation plots
8	Model Execution time was not calculated	Model execution Time was calculated and plotted
9	Model Evaluation Metrics were only Precision and Accuracy	Accuracy, Precision, F1 Score, Execution Time and Recall were used
10	AUC Score or ROC Curve is not plotted	AUC Score was calculated, plotted ROC Curve
11	Ensemble Modeling/ Deep learning methods were not used	Ensemble Modeling/ Deep learning methods were used

12. Conclusion

In this milestone-3, we used template code to preprocess the data by removing all redundant and missing values, applied label encoder to convert all categorical features into numerical performed exploratory data analysis, selected important features and applied cross-validation method on various machine learning classifiers to predict whether a job posting is genuine or fake. We developed and trained the model, evaluated its performance, and visualized the ROC curve. Our best-performing model was XGBoost, which had an accuracy of 0.98 and an AUC score of 0.99. We can use this model to predict whether a job posting is genuine or fake with high accuracy.

By applying template-2 code, we have learnt about ensemble modelling and the factors that affect the voting classifiers and also learnt on how to apply voting classifier-max vote or average to other classifier models like Random Forest, Logistic Regression, KNN and Decision Tree. The Decision Tree plot helped to understand how each feature is correlated with other features.

By the efforts from Natural Language processing and Deep learning techniques, we have cleaned the dataset for textual processing and applied LSTM technique to the textual dataset. The Model training and evaluation helped to understand various hyper-parameters involved in implementing a deep learning neural network algorithms to a dataset. Finally, the visualization gave a clear understanding of the validation of training and test datasets. Post which the model evaluation was done on different evaluation metrics like Precision, Accuracy, F1-score and Recall.

13. References

- Ahmed, H., Traore, I., & Saad, S. (2017). Detection of online fake news using n-gram analysis and machine learning techniques. In International conference on intelligent, secure, and dependable systems in distributed and cloud environments, 127-138.
- Allcott, H., & Gentzkow, M. (2017). Social media and fake news in the 2016 election. Journal of economic perspectives, 31(2), 211-36.
- Gupta, D., Khanna, A., & Gupta, V. (2021). Job Fraud Detection using Machine Learning. International Journal of Computer Applications, 180(21), 13-18.
- Srivastava, N., & Gupta, R. (2021). Fake Job Posting Detection using Machine Learning. International Journal of Advanced Research in Computer Science, 12(4), 6-11.
- Rana, M., & Sharma, A. (2021). A Comparative Study of Machine Learning and Deep Learning Approaches for Fake Job Posting Detection. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 6(1), 72-78.
- Khan, S. H., Qayyum, F., & Batool, R. (2020). Fake Job Posting Detection using Machine Learning Techniques. International Journal of Computer Science and Mobile Computing, 9(8), 81-86.
- Kumar, A., & Dwivedi, A. K. (2020). Fake Job Posting Detection using Machine Learning: A Comparative Study. International Journal of Emerging Technologies in Engineering Research, 8(12), 12-17.
- Rani, M. S., & Kumar, S. (2021). Job Fraud Detection using Machine Learning Techniques. International Journal of Advanced Science and Technology, 30(1), 609-614.
- https://www.kaggle.com/code/shivamburnwal/nlp-98-acc-eda-with-model-using-spacy-pipeline
- https://www.kaggle.com/code/madz2000/text-classification-using-keras-nb-97-accuracy
- How to Choose a Feature Selection Method For Machine Learning:
- Feature Selection Techniques in Machine Learning
- Fake Job Posting Detection and Getting Useful Insights from the Job Postings
- https://github.com/sharad18/Fake_Job_Posting/blob/master/Data%20Cleaning%20%26%20EDA.ipynb
- https://www.kaggle.com/code/shivamburnwal/nlp-98-acc-eda-with-model-using-spacy-pipeline