Fraudulent Job Posting Detection

Performing exploratory data analysis to understand and analyze fraudulent job postings on various job boards using advanced machine learning techniques.

Table of contents

Authors

1.	Anirudh Mhaske	50604524 amhaske2@buffalo.edu
2.	Dhiraj Patil	50604210 dhirajsu@buffalo.edu
3.	Vidit Rajesh Prabhu	50600684 viditraj@buffalo.edu
4.	Hemasree Pujari	50610526 hemasree@buffalo.edu

Problem Statement Exploration

Title

Fraudulent job posting detection on online job boards using various machine learning techniques.

Problem Statement

The increase in fraudulent job postings has become a significant concern in the ever-evolving online job market leading to dire consequences for hopeful job seekers such as emotional distress, financial loss, and even reports of identity theft concerns. Our goal lies in effectively identifying and eventually mitigating these deceptive job postings to protect job seekers and maintain the integrity of online job portals.

Significance

This dataset is really important because it tackles the growing problem of fake job postings in the online job market. These scams don't just waste people's time, they may cause serious harm like emotional distress, financial loss, and even identity theft. By finding and reducing these fraudulent activities we can:

1. **Build trust** in online job portals.
2. **Protect job seekers** from being scammed.
3. Keep job platforms **reliable and credible**.
4. Set the stage for **advanced fraud detection research** beyond just job postings.

Potential Impact

Using this dataset can help us:

- Improve **early warning systems** for spotting fake job ads, making job portals more secure.
- Use **data insights** to build smarter filters and detection methods.
- Increase **user confidence** in online job boards.
- Apply the learnings to other areas where fraud happens like real estate listings or online shopping.

By addressing fake job listings this research not only helps create a safer job hunting experience but also lays the groundwork for stronger fraud detection tools that can be used across various industries.

Data Sources

Source

The dataset used for this project is titled “Real / Fake Job Posting Prediction”. This dataset has been sourced from Kaggle.

Link for the Dataset - https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction

Description

It comprises around 18,000 job descriptions. The data includes both textual information and meta information about the jobs. The dataset can be used to train classification models that can identify fake job descriptions.

Our dataset contains the following features, that are, location, company profile, department, description, requirements, benefits, salary range, employment type, required experience, required education, industry, and functions.

This dataset acts as an extremely valuable source for several key pieces of information. It allows us to process a classification model using text-based and meta-features to predict whether a job description is fraudulent or genuine. Additionally, it also helps us to identify main features such as specific words, objects, and phrases that are commonly witnessed in fraudulent job postings. Furthermore, this dataset creates a pathway for contextual embedding models which can be run to find out the most similar job descriptions. Finally, we will be able to perform exploratory data analysis on the dataset which will allow us to reveal insights and patterns.

Additional Information

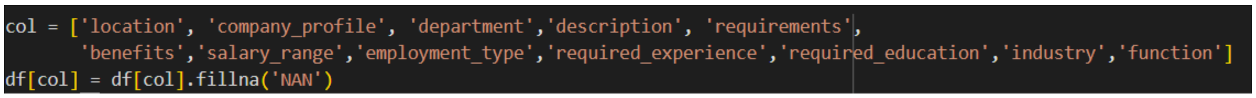
The dataset was originally prepared by the Laboratory of Information & Communication Systems Security at The University of the Aegean.

Data Cleaning and Processing

Data cleaning and processing steps are crucial for converting raw data into processed data which can be used for analysis. The data cleaning step involves identifying and correcting errors, handling missing values, removing duplicating values, and fixing inconsistencies in the dataset. Data processing then transforms this cleaned data into a suitable format which can then be used for further analysis by machine learning models.

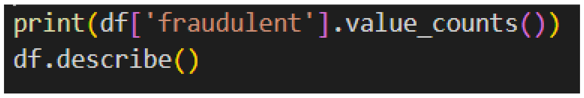
Handling missing entries

Our dataset contained approximately more than 70k missing values, which would not be beneficial for our model to train on. To handle these missing entries we substitute with ‘NaN’ object values which can then easily be identified and processed by our model.

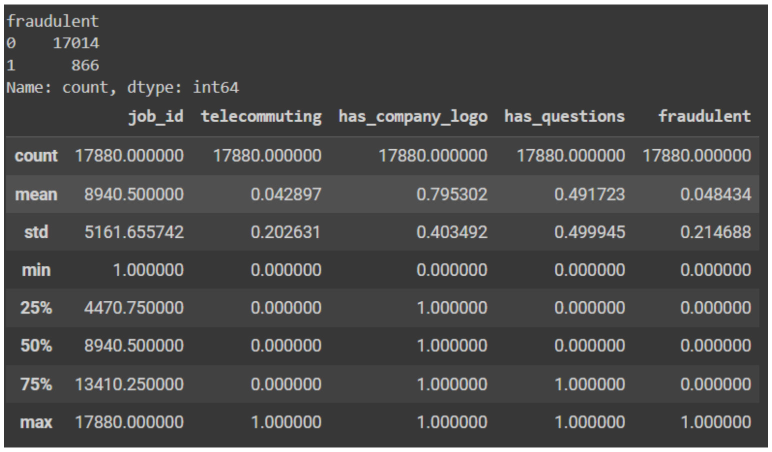


```
col = ['location', 'company_profile', 'department','description', 'requirements',
       'benefits','salary_range','employment_type','required_experience','required_education','industry','function']
df[col] = df[col].fillna('NAN')
```

Statistical Ideation



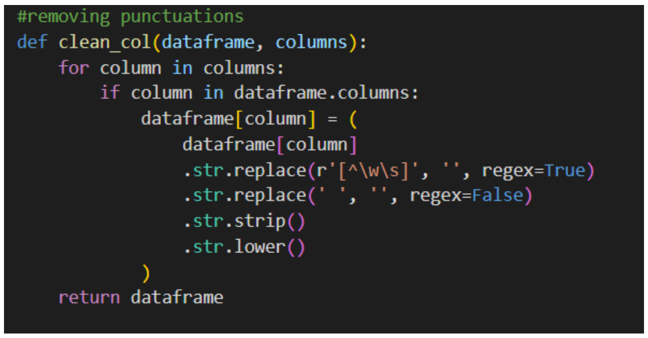
```
print(df['fraudulent'].value_counts())
df.describe()
```



```
fraudulent
0    17014
1      866
Name: count, dtype: int64
            job_id  telecommuting  has_company_logo  has_questions   fraudulent
count  17880.000000   17880.000000      17880.000000   17880.000000  17880.000000
mean    8940.500000       0.042897          0.795302       0.491723      0.048434
std     5161.655742       0.202631          0.403492       0.499945      0.214688
min        1.000000       0.000000          0.000000       0.000000      0.000000
25%     4470.750000       0.000000          1.000000       0.000000      0.000000
50%     8940.500000       0.000000          1.000000       0.000000      0.000000
75%    13410.250000       0.000000          1.000000       1.000000      0.000000
max    17880.000000       1.000000          1.000000       1.000000      1.000000
```

The dataset has 17,014 real job postings and 866 fake ones. So only about 5% of the listings are fraudulent. This shows there’s an imbalance which is important to consider when building predictive models for detecting fake job postings.

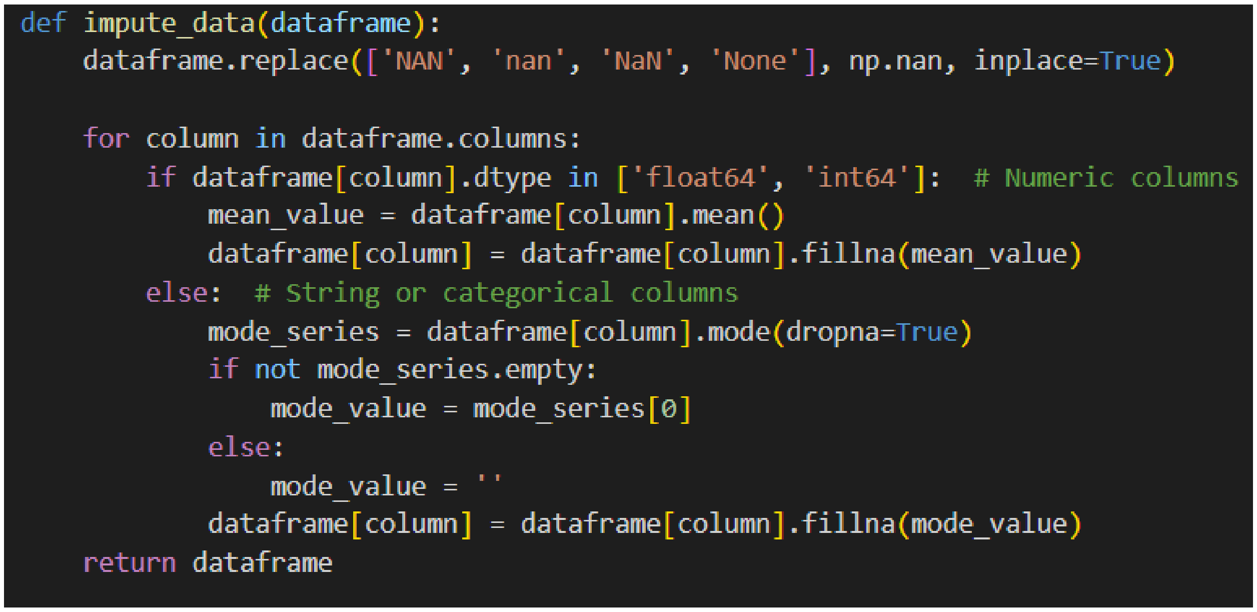
Data Standardization



```
#removing punctuations
def clean_col(dataframe, columns):
    for column in columns:
        if column in dataframe.columns:
            dataframe[column] = (
                dataframe[column]
                .str.replace(r'[^\w\s]', '', regex=True)
                .str.replace(' ', '', regex=False)
                .str.strip()
                .str.lower()
            )
    return dataframe
```

The clean_col function was used to remove punctuation, spaces, and convert all text to lowercase in selected columns of the dataset (like 'title', 'location', 'company_profile', etc). This helps to standardize the data and makes it easier to process for further analysis or model building. In the cleaned data the textual information is now continuous without special characters or unnecessary spaces. This step improves the quality and consistency of the data for model training or text based analysis.

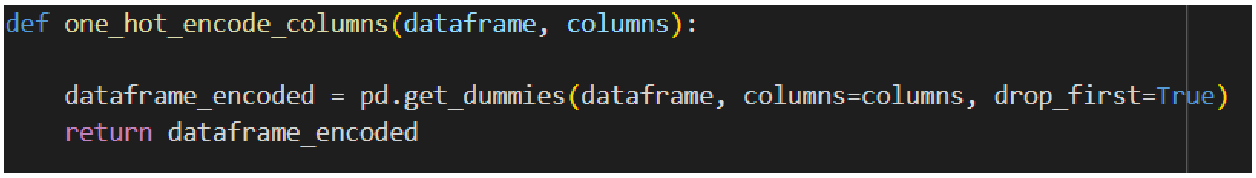
Imputing Missing values



```
def impute_data(dataframe):
    dataframe.replace(['NAN', 'nan', 'NaN', 'None'], np.nan, inplace=True)

    for column in dataframe.columns:
        if dataframe[column].dtype in ['float64', 'int64']:  # Numeric columns
            mean_value = dataframe[column].mean()
            dataframe[column] = dataframe[column].fillna(mean_value)
        else:  # String or categorical columns
            mode_series = dataframe[column].mode(dropna=True)
            if not mode_series.empty:
                mode_value = mode_series[0]
            else:
                mode_value = ''
            dataframe[column] = dataframe[column].fillna(mode_value)
    return dataframe
```

In this step we dealt with missing values to clean up the dataset for further analysis. The function replaced common placeholders like NAN and None with proper NaN values. For numeric columns we filled in missing data with the average of that column and for non-numeric ones we used the most frequently occurring value (mode) to make sure that we don't lose important data and can move ahead with accurate analysis.

Categorizing Dataset

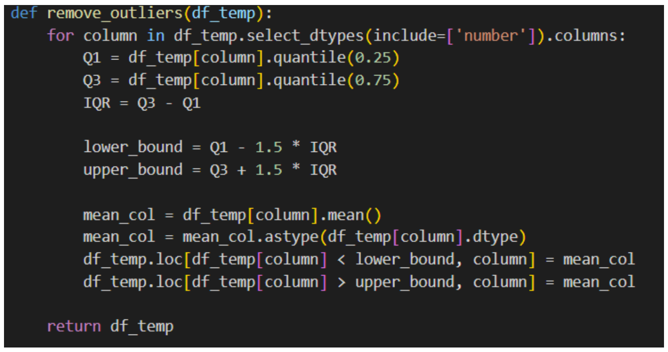


```
def one_hot_encode_columns(dataframe, columns):

    dataframe_encoded = pd.get_dummies(dataframe, columns=columns, drop_first=True)
    return dataframe_encoded
```

In this step we're using the one hot encoding function to convert the categorical columns in the dataset into a numerical format. This is important because machine learning models work better with numbers than text. The function takes in a list of columns like job title, location, department, and more and converts each unique value in these columns into separate binary (0 or 1) columns.

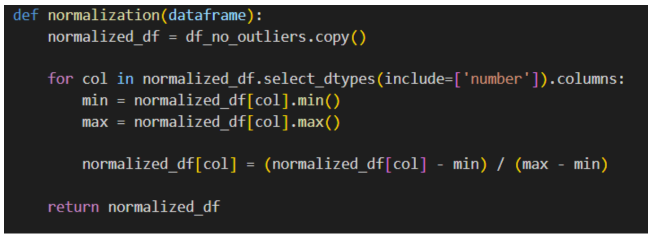
Exploratory Data Analysis

Handling Outliers


```
def remove_outliers(df_temp):
    for column in df_temp.select_dtypes(include=['number']).columns:
        Q1 = df_temp[column].quantile(0.25)
        Q3 = df_temp[column].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        mean_col = df_temp[column].mean()
        mean_col = mean_col.astype(df_temp[column].dtype)
        df_temp.loc[df_temp[column] < lower_bound, column] = mean_col
        df_temp.loc[df_temp[column] > upper_bound, column] = mean_col

    return df_temp
```

This function remove_outliers handles outliers in the dataset without dropping them. Instead, we replace them with the column's average. For each numeric column, the function calculates the Interquartile Range which is the range between the 25th and 75th percentiles. Any value below the lower bound (Q1 - 1.5 * IQR) or above the upper bound (Q3 + 1.5 * IQR) is flagged as an outlier. But instead of removing these outliers, we replace them with the column's mean value. This way we don't lose data but we still handle those extreme values that might affect our analysis.

Normalizing Values


```
def normalization(dataframe):
    normalized_df = df_no_outliers.copy()

    for col in normalized_df.select_dtypes(include=['number']).columns:
        min = normalized_df[col].min()
        max = normalized_df[col].max()

        normalized_df[col] = (normalized_df[col] - min) / (max - min)

    return normalized_df
```

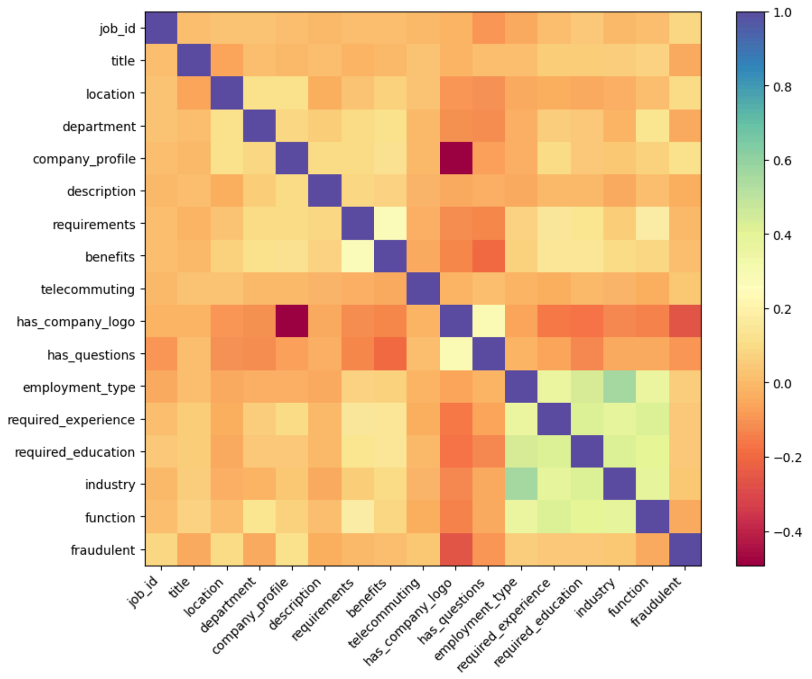
In this normalization function, we normalize the numeric columns to bring all values between 0 and 1. This ensures that all features are on the same scale which helps improve model performance. The function calculates the minimum and maximum values for each column and then scales the values accordingly.

Label Encoding

```python
def label_encode(dataframe, columns):
    dataframe_encoded = dataframe.copy()
    lable = LabelEncoder()

    for column in columns:
        if column in dataframe_encoded.columns:
            dataframe_encoded[column] = lable.fit_transform(dataframe_encoded[column])

    return dataframe_encoded
```

This function applies label encoding to the specified categorical columns in the dataframe for transforming them into numerical values for easier processing in machine learning models. It iterates over the given columns for fitting and transforming each one using LabelEncoder.

Generating Correlation Matrix



This heatmap shows how features in job postings are related. Blue indicates strong positive, red shows negative, and yellow means no correlation. Fraudulent jobs tend not to mention employment type, experience, or education. Jobs with company logos usually don’t offer remote options. There's a positive link between industry, education, and experience, meaning jobs that mention one often include the others. This helps in spotting patterns.

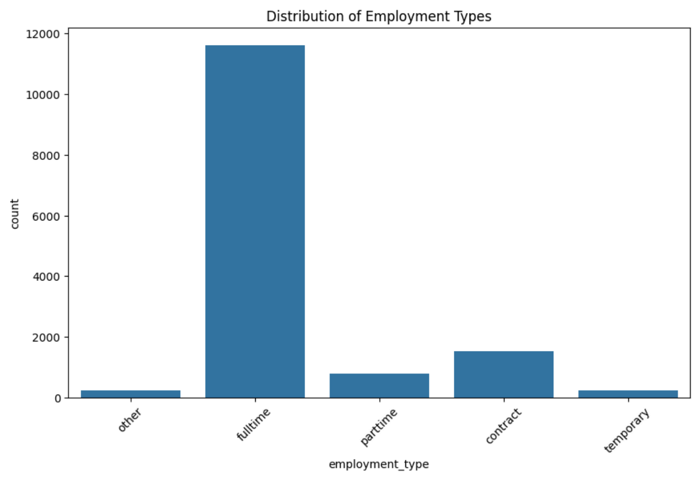
Dropping Irrelevant Features

```
df_dropped_col = df.drop(columns=['job_id', 'department', 'benefits'])
df_encoded_ohe = df.drop(columns=['job_id', 'department', 'benefits'])
```

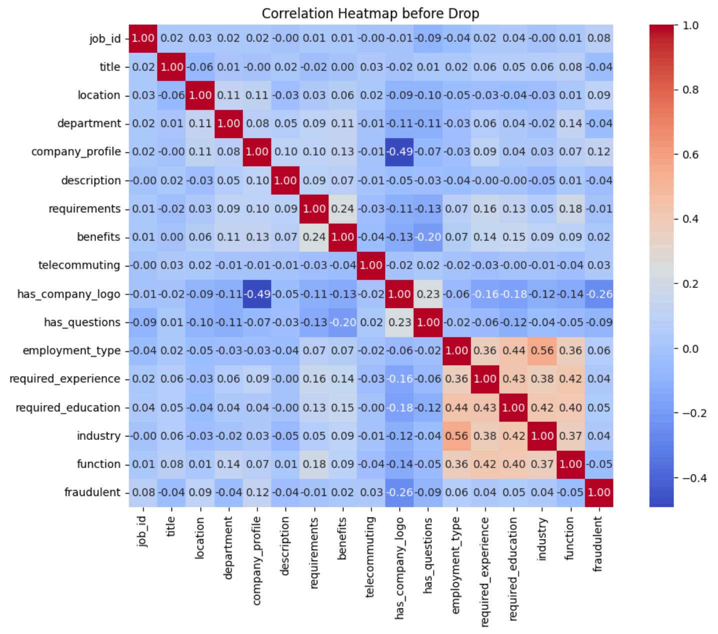
To reduce noise and improve the performance of the model we removed columns that do not contribute meaningful information so that we can focus on only relevant features.

Data Visualizations

For the following graph, we are visualizing the distribution of employment types based on the count of total number of job postings present for each job role type. This visualization allows us to understand which employment type will majorly affect our model.



For the following visualization, we have generated a correlation heatmap utilizing the seaborn library, allowing us to understand how the important features are correlated to each other. With the help of this heatmap, we can identify the most relevant features which can then be further used for our classification model.



Splitting Dataset

```
X = df_encoded_ohe.drop(columns=['fraudulent'])  # Features
y = df_encoded_ohe['fraudulent']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("Training features:", X_train.shape)
print("Test features:", X_test.shape)
print("Training target:", y_train.shape)
print("Test target:", y_test.shape)
```

```
Training features: (14304, 14)
Test features: (3576, 14)
Training target: (14304,)
Test target: (3576,)
```

We are splitting the dataset into features (X) and the target (y), where y represents if a job is fraudulent or not. Then we perform an 80-20 split using train_test_split() which means 80% of the data goes into training the model and 20% is used for testing. By setting random_state=42 we ensure the split is consistent every time we run the code.

Machine Learning algorithms:

1. Decision Tree

Decision Tree is a straightforward yet powerful machine learning model used for classification and regression. It splits data based on feature values, forming a tree-like structure where each branch represents a decision. The model keeps dividing the data into smaller subsets until it reaches a final prediction. It's like asking a series of yes/no questions to arrive at a decision.

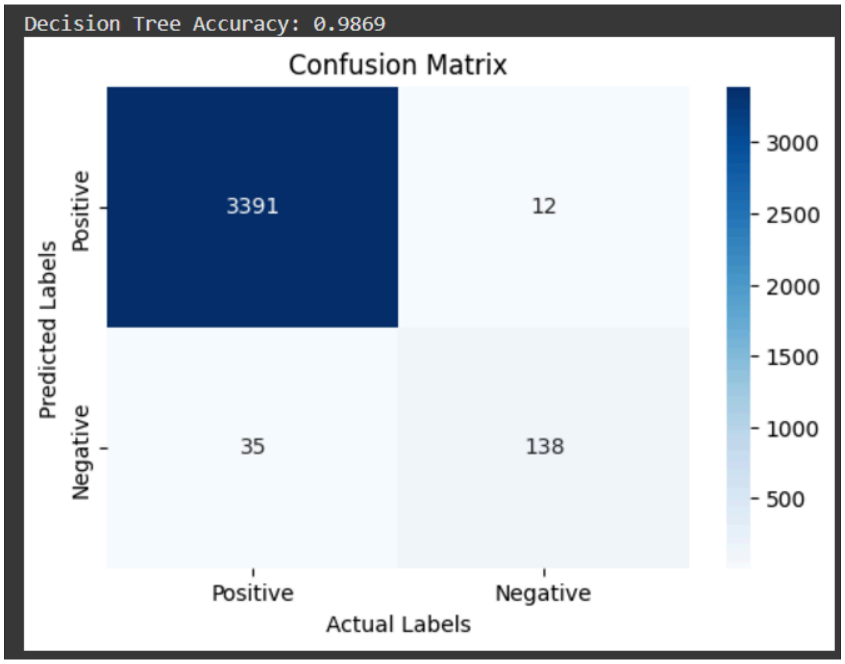
Reason for Choosing:
Decision Trees are easy to understand and visualize which makes them helpful for exploring which features are most important in predicting fraudulent job postings. They offer clear explanations for why a particular prediction is made making them great for initial analysis and understanding key factors in dataset.

Training and Tuning

- **Training**: For Decision Tree model we used DecisionTreeClassifier() from Scikit-Learn. model was trained using training dataset (X_train and y_train). During training model learned to split dataset into subsets based on features that best differentiate between fraudulent and legitimate job postings.
- **Tuning**: In this particular implementation, we did not explicitly tune hyperparameters.

Evaluation and Performance Metrics

- **Accuracy**: accuracy of Decision Tree model was **0.9869** which indicates that model correctly predicted around 98.7% of job postings as either fraudulent or legitimate. High accuracy suggests that model performs well in most cases.
- **Confusion Matrix**: confusion matrix, as shown in provided image, gives a detailed breakdown of model's predictions:


Decision Tree Accuracy: 0.9869

    - **True Positives (TP)**: 3391 job postings were correctly predicted as legitimate.
    - **True Negatives (TN)**: 138 job postings were correctly predicted as fraudulent.
    - **False Positives (FP)**: 12 legitimate postings were incorrectly predicted as fraudulent.
    - **False Negatives (FN)**: 35 fraudulent postings were incorrectly predicted as legitimate.
- **Insights**:
    - **low number of false positives** (12) suggests that model does not often mistakenly classify legitimate job postings as fraudulent, which is important for minimizing disruption to genuine job posters.
    - **false negatives** (35), while relatively low, indicate that some fraudulent postings were missed by model. This could be a concern because failing to identify fraudulent postings may expose job seekers to potential risks.
    - high number of **true positives and true negatives** indicates that model generally understands features that distinguish between fraudulent and legitimate postings.

Effectiveness in Addressing Problem Statement

Decision Tree model is highly effective in providing transparency into which features are most impactful in predicting fraud. This aligns well with our problem statement which aims to not only detect fraudulent postings but also understand what makes these postings suspicious. model's ability to visually represent decision paths is a significant advantage for this type of analysis.

However Decision Tree has a tendency to **overfit** especially if no constraints (like limiting max_depth) are applied. This means that while model performs very well on training data, it might struggle to maintain this level of performance when exposed to new unseen data.

In this case high accuracy of 98.7% is promising but it also indicates that **overfitting might be occurring** as real-world datasets often have more noise and unseen variations. Using ensemble methods like **Random Forest** can mitigate these issues by aggregating multiple decision trees thus improving generalizability.

2.Gradient Boosting Classifier

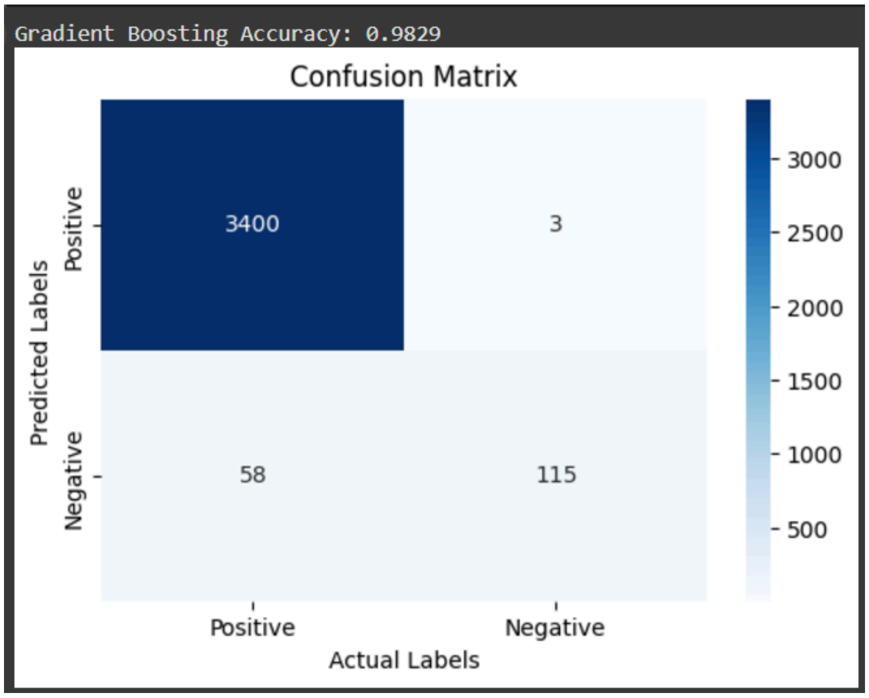
Gradient Boosting is an advanced ensemble technique that builds decision trees sequentially. Each new tree is trained to correct mistakes made by previous trees which gradually improves model's accuracy. It effectively reduces errors by focusing on residuals from previous predictions resulting in a model that can capture intricate patterns in data.

Reason for Choosing:
Gradient Boosting was chosen because of its ability to handle complex classification tasks with high accuracy. By learning from its own mistakes step-by-step it becomes very effective at picking up subtle patterns in data. This approach is particularly good for reducing false positives which is crucial for ensuring that legitimate job postings are not mistakenly flagged as fraudulent.

Training and Tuning:
For Gradient Boosting, we used GradientBoostingClassifier() from Scikit-Learn and trained it using training dataset (X_train and y_train). model was trained with default hyperparameters focusing on sequentially improving weak learners (decision trees) to minimize errors. We chose Gradient Boosting because it is effective at handling complex relationships in data by learning from mistakes in each iteration.

Evaluation and Performance Metrics:
model was evaluated on test dataset (X_test). Here are key metrics:

- **Accuracy**: **98.29%** — model demonstrated strong predictive power, correctly classifying almost all job postings.
- **Confusion Matrix**:


```
Gradient Boosting Accuracy: 0.9829
```

- **True Positives (TP)**: 3400 job postings correctly classified as legitimate.
- **True Negatives (TN)**: 115 postings correctly classified as fraudulent.
- **False Positives (FP)**: 3 legitimate postings were incorrectly labeled as fraudulent.
- **False Negatives (FN)**: 58 fraudulent postings were incorrectly labeled as legitimate.

Effectiveness in Addressing Problem Statement:
Gradient Boosting model proved to be highly effective in detecting fraudulent job postings with an accuracy of 98.29%. low number of false positives is particularly beneficial as it ensures that legitimate job postings are not mistakenly flagged thereby maintaining trust among job posters. Additionally reduction in false negatives (compared to other models) means that more fraudulent postings are correctly identified, reducing risk to job seekers. Gradient Boosting's iterative learning approach allowed model to capture complex patterns in dataset, making it more capable of distinguishing subtle indicators of fraud. However it is computationally intensive which could be a consideration for large-scale applications. Despite this, its high performance makes it a strong choice for mitigating risks associated with fraudulent job postings and protecting job seekers.

3. Random Forest

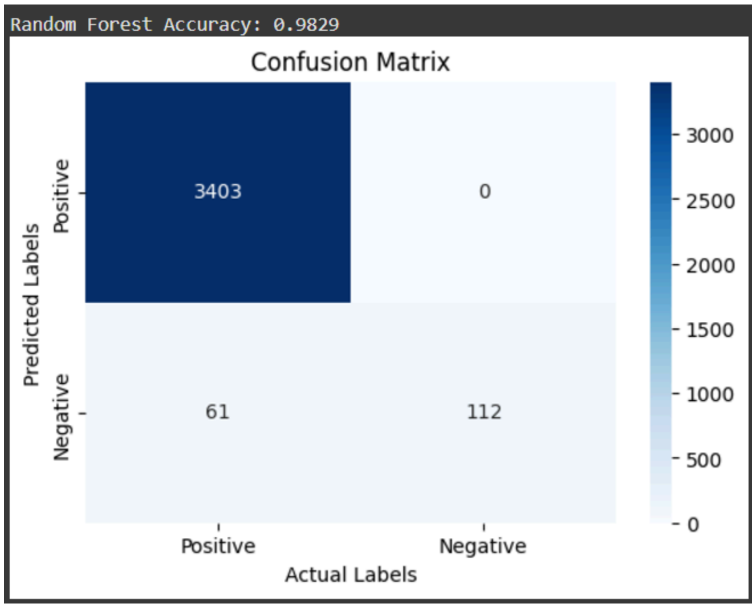
Random Forest is an ensemble method that creates a collection of decision trees each trained on different random samples of data. Model averages predictions from all these trees to provide a more accurate and stable result. This method helps reduce overfitting and makes model more robust by taking multiple perspectives on data.

Reason for Choosing:
Random Forest was chosen because it can handle complex datasets effectively while reducing overfitting issues that often occur with single Decision Trees. It also provides insights into which features are most influential, which is valuable for understanding what drives fraudulent job postings.

Training and Tuning:
For Random Forest model, we used RandomForestClassifier() from Scikit-Learn and trained it using training dataset (X_train and y_train). Random Forest is an ensemble method that builds multiple decision trees using different parts of dataset and averages their predictions for better accuracy and robustness. We used default hyperparameters for this initial analysis.

Evaluation and Performance Metrics:
model was evaluated on test dataset (X_test). Here are key metrics:

- **Accuracy**: **98.29%** — model showed excellent performance in classifying job postings correctly.
- **Confusion Matrix**:



```
Random Forest Accuracy: 0.9829
```

  - **True Positives (TP)**: 3403 job postings were correctly classified as legitimate.
  - **True Negatives (TN)**: 112 postings were correctly classified as fraudulent.
  - **False Positives (FP)**: 0 legitimate postings were incorrectly classified as fraudulent.
  - **False Negatives (FN)**: 61 fraudulent postings were missed.

Effectiveness in Addressing Problem Statement:
Random Forest model effectively detected fraudulent job postings with an accuracy of 98.29%. fact that there were no false positives indicates that model is highly reliable in ensuring legitimate job postings are not flagged incorrectly, which is crucial for maintaining trust with job seekers and job posters alike.

61 fraudulent postings were missed, highlighting that there is still some risk for job seekers. Random Forest's strength lies in its ability to reduce overfitting compared to individual decision trees while maintaining interpretability regarding feature importance. This model's robustness and accuracy make it a strong contender for identifying fraudulent job postings, though further tuning could potentially reduce number of false negatives and increase safety for job seekers.

4. Logistic Regression

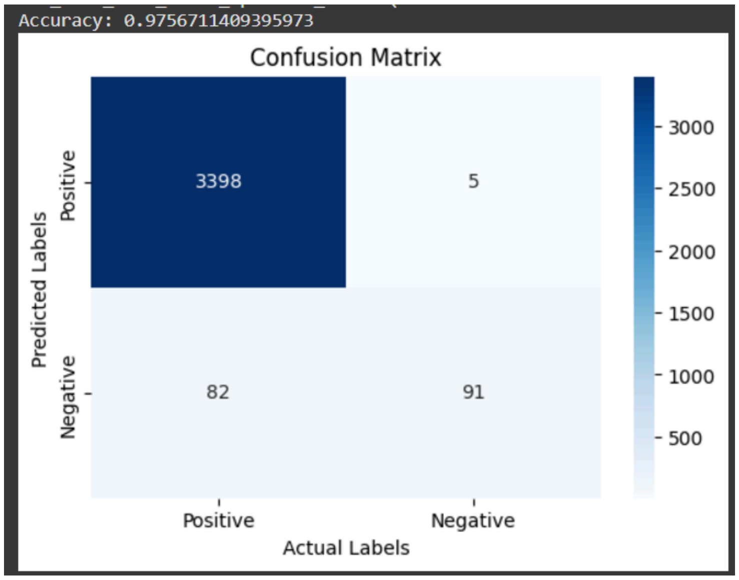
Logistic Regression is a simple classification model that estimates probability that an instance belongs to a particular class. It uses a linear combination of features and applies a logistic function to output values between 0 and 1, representing probabilities. This model is useful for binary classification like predicting whether a job posting is fraudulent or not.

Reason for Choosing:
Logistic Regression was selected as a baseline model due to its simplicity and interpretability. It provides a quick way to see how features relate to likelihood of fraud offering coefficients that tell us how much each feature influences outcome. It's a great starting point to understand relationships in data.

Training and Tuning:
For Logistic Regression we used training dataset (X_train and y_train) and increased number of iterations (max_iter=1000) to ensure model had enough time to converge. We set random_state to 42 to guarantee consistent results across different runs. Logistic Regression was chosen as a baseline due to its simplicity and ease of interpretation, which makes it suitable for understanding relationships between features and identifying factors influencing fraudulent job postings. While we didn't perform exhaustive hyperparameter tuning, potential parameters we could have optimized include:

Evaluation and Performance Metrics:
The model was evaluated on test set (X_test). Here are key metrics:

- **Accuracy**: **97.57%** — model accurately classified vast majority of job postings as either legitimate or fraudulent.
- **Confusion Matrix**:


```
Accuracy: 0.9756711409395973
```

- **True Positives (TP)**: 3398 job postings correctly classified as legitimate.
- **True Negatives (TN)**: 91 job postings correctly classified as fraudulent.
- **False Positives (FP)**: 5 legitimate postings were incorrectly labeled as fraudulent.
- **False Negatives (FN)**: 82 fraudulent postings were incorrectly labeled as legitimate.

Effectiveness in Addressing Problem Statement:

Logistic Regression model has shown good accuracy, with an accuracy of 97.57%. low number of false positives is a positive outcome, as it ensures that genuine job postings are rarely mislabeled, maintaining trust with legitimate employers. However model had 82 false negatives, which means some fraudulent postings were missed, posing a risk to job seekers. This is particularly concerning given potential harm fraudulent job postings can cause.. To address these shortcomings, more advanced models like Random Forest or Gradient Boosting could be used to reduce false negatives and improve overall detection capabilities.

5. Support Vector Machine (SVM)

SVM is a powerful model that finds best boundary (hyperplane) to separate different classes in data. When data isn't easily separable in its original form SVM uses a technique called kernel trick to transform it into a higher-dimensional space where a linear boundary can be found. goal is to maximize margin between classes, which helps improve model's ability to generalize.

Reason for Choosing:
SVM is used best for cases where data has many features and classes are not linearly separable. It is effective at creating clear decision boundaries to classify job postings even when underlying relationships are complex. This makes it a strong choice for accurately distinguishing between fraudulent and legitimate job postings.

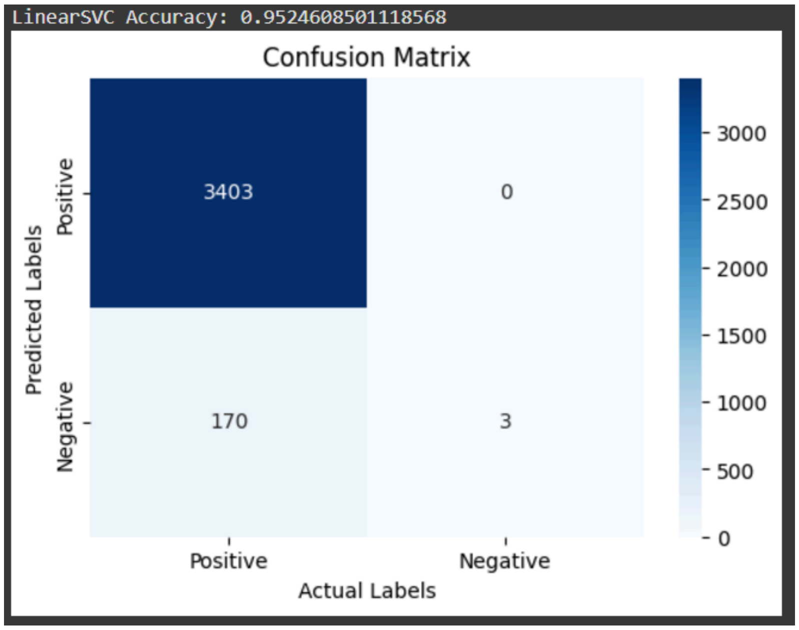
Training and Tuning:
For LinearSVC, we used LinearSVC() model from Scikit-Learn and trained it on training dataset (X_train and y_train). max_iter parameter was set to 10,000 to allow model sufficient iterations to converge, and random_state was set to 42 to ensure reproducibility of results. LinearSVC was chosen because of its ability to efficiently handle large-scale datasets and produce a linear decision boundary. Although we did not perform extensive hyperparameter tuning for this model, common parameters to optimize include:

- **C**: Regularization strength that helps control overfitting.
- **tol**: tolerance for stopping criteria, which impacts model convergence.

Evaluation and Performance Metrics:

model's performance was evaluated on test dataset (X_test). Below are key metrics:

- **Accuracy**: **95.25%** — Model was able to correctly classify most job postings as either fraudulent or legitimate.
- **Confusion Matrix**:


```
LinearSVC Accuracy: 0.9524608501118568
```

  - **True Positives (TP)**: 3403 job postings were correctly classified as legitimate.
  - **True Negatives (TN)**: 3 postings were correctly classified as fraudulent.
  - **False Positives (FP)**: 0 legitimate postings were incorrectly classified as fraudulent.
  - **False Negatives (FN)**: 170 fraudulent postings were missed.

Effectiveness in Addressing Problem Statement:
LinearSVC achieved an accuracy of 95.25%, which is reasonably high, but its performance had some shortcomings, especially in detecting fraudulent postings. model's high number of **false negatives** (170) indicates that a significant portion of fraudulent postings went undetected. This is concerning given potential harm such postings can cause to job seekers. On positive side, there were no false positives,

which helps maintain trust of legitimate job posters. LinearSVC's linear nature allowed for efficient training and scalability, but it may not have captured more complex patterns present in fraudulent postings compared to non-linear models.

References

1. https://www.kdnuggets.com/2023/08/7-steps-mastering-data-cleaning-preprocessing-techniques.html
2. https://matplotlib.org/stable/users/index.html
3. https://seaborn.pydata.org/tutorial/function_overview.html
4. https://scikit-learn.org/1.5/modules/tree.html
5. https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.RandomForestClassifier.html
6. https://scikit-learn.org/dev/modules/generated/sklearn.linear_model.LogisticRegression.html
7. https://scikit-learn.org/1.5/modules/svm.html
8. https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html