Data Scientist Job Change Analysis Project

Introduction:

Employee retention is an important factor which measures the growth and success of the Company. This project aims to analyze a dataset of a company to identify key factors which are responsible for data scientist employee finding new jobs or working for the same company after a training program. By making use of various data analysis techniques, we can draw meaningful insights that could assist recruiters in maintaining employees and improve their decision making process.

Objective:

The primary objectives of this project are:
To explore and understand the job change dataset.
To perform data preprocessing to handle missing or incorrect values and to make the dataset ready for analysis.
To analyse and visualise the dataset using its various factors to find meaningful insights related to the dataset.
To build a machine learning model that can classify whether or not an employee is to change jobs based on the provided reasons and situations.

Dataset Overview:

The dataset consists of 14 columns in the training set and 13 columns in the test set, covering information about city development index, gender, education, major discipline, company size, type, and whether the candidate is actively looking for a job change (target).

Key Features in the Dataset:

- **enrollee_id:** Unique ID for each candidate.
- **city:** City code.
- **city_development_index:** Development index of the city (scaled between 0 and 1).
- **gender:** Gender of the candidate.
- **relevant_experience:** Whether the candidate has relevant work experience.
- **enrolled_university:** Type of University course enrolled (if any).
- **education_level:** Highest level of education achieved.

- **major_discipline:** Major discipline of education.
- **experience:** Total work experience of the candidate (in years).
- **company_size:** Number of employees in the candidate's current company.
- **company_type:** Type of current employer.
- **last_new_job:** Time since the candidate's last job change (in years).
- **training_hours:** Number of hours spent in training by the candidate.
- **target:** 0 – Not looking for a job change, 1 – Looking for a job change.

1. Data Collection and Loading:

We started by importing the necessary libraries and loading the dataset into pandas DataFrames for both the training and test datasets.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from lazypredict.Supervised import LazyClassifier, LazyRegressor

from sklearn.preprocessing import LabelEncoder, QuantileTransformer, StandardScaler, MinMaxScaler

from sklearn.model_selection import train_test_split,cross_val_score

from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, classification_report, RocCurveDisplay

from sklearn.model_selection import GridSearchCV, cross_validate

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier


# Loading the dataset

train = pd.read_csv('train.csv')
```

```
test = pd.read_csv('test.csv')
```

Basic Overview:

- **Training set shape:** 19,158 rows × 14 columns
- **Test set shape:** 2,129 rows × 13 columns

```
# Displaying the first few rows of the datasets

train.head()

test.head()
```

The training set contains one additional column (target) which is not present in the test set. This is the column we will be predicting.

2. Data Preprocessing

2.1 Handling Missing Values:

We performed an in-depth analysis to identify missing values within the dataset.

```
# Checking missing values for each column

train.info()

test.info()
```

Key Observations:

- Some columns like gender, enrolled_university, education_level, major_discipline, experience, company_size, and company_type had missing values.

To handle these, we opted to fill missing values based on logical assumptions:

- **Gender** and **company type**: Replaced with 'Not Known'.
- **Company size**: Mode imputation, replacing NaN with the most frequent size category.
- **Experience and last_new_job**: Filled with median or average values.

```
# Filling missing values

train['gender'] = train['gender'].fillna('Not Known')

test['gender'] = test['gender'].fillna('Not Known')
```

```
train['company_size'] = train['company_size'].fillna('50-99')  # Mode imputation

test['company_size'] = test['company_size'].fillna('50-99')


train['last_new_job'] = train['last_new_job'].fillna('1')

test['last_new_job'] = test['last_new_job'].fillna('1')
```

2.2 Transforming Categorical Data:

For columns like experience and last_new_job, which contain non-numeric values such as '>20' and 'never', we replaced them with appropriate numeric equivalents.

```
# Replacing '>20' with 20 and '<1' with 1 for experience

train['experience'] = train['experience'].replace({'>20': 20, '<1': 1}).astype(int)

test['experience'] = test['experience'].replace({'>20': 20, '<1': 1}).astype(int)


# Converting 'never' to 0 and '>4' to 5 in last_new_job

train['last_new_job'] = train['last_new_job'].replace({'never': 0, '>4': 5}).astype(int)

test['last_new_job'] = test['last_new_job'].replace({'never': 0, '>4': 5}).astype(int)
```

3. Exploratory Data Analysis (EDA):

3.1 Unique Values in the Dataset

The dataset contains a range of categorical and numerical variables, each with varying levels of uniqueness.

```
train.nunique()
```

Column	Unique Values
`enrollee_id`	19,158
`city`	123
`city_development_index`	93
`gender`	4
`relevent_experience`	2
`enrolled_university`	4
`education_level`	6
`major_discipline`	7
`experience`	20
`company_size`	8
`company_type`	6
`last_new_job`	6
`training_hours`	241
`target`	2

3.2 Analysis of Average Training Hours by Education Level

We examined the relationship between education_level and training_hours to see how much time candidates with different educational backgrounds spend in training.

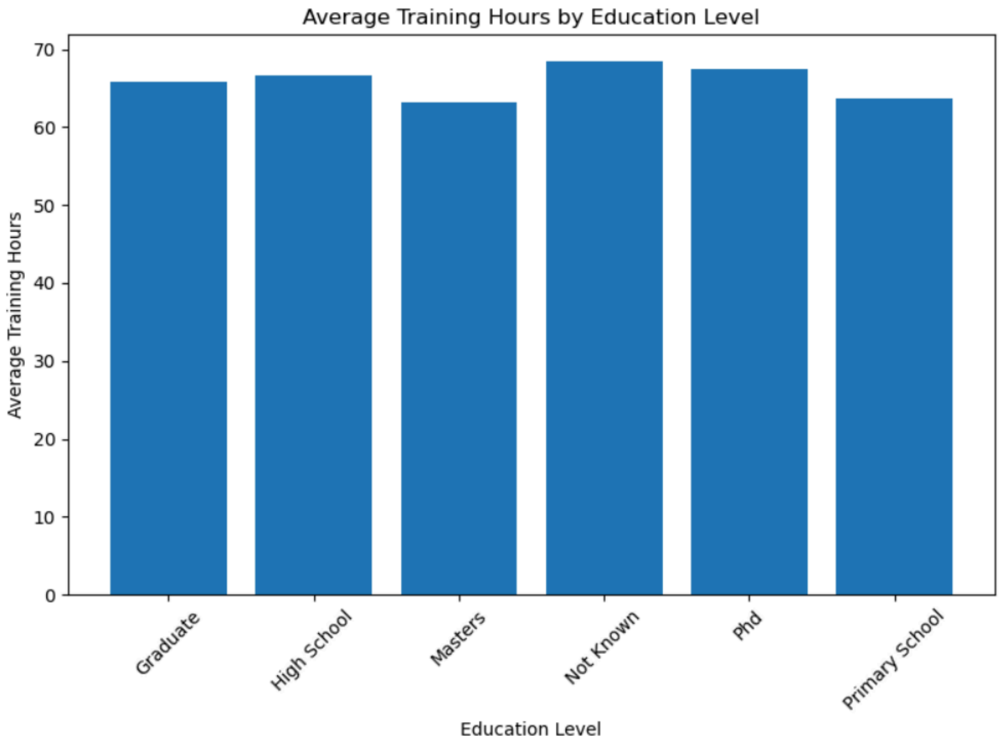
```
education_training_hours = train.groupby('education_level')['training_hours'].mean()
education_training_hours
```

Education Level	Avg. Training Hours
Graduate	65.77
High School	66.68
Masters	63.27
Not Known	68.45
PhD	67.52
Primary School	63.62

A bar chart was created to display the average training hours across different education levels.

```
plt.figure(figsize=(8,6))
plt.bar(education_training_hours.index, education_training_hours.values)
plt.xlabel('Education Level')
plt.ylabel('Average Training Hours')
plt.title('Average Training Hours by Education Level')
plt.xticks(rotation=45)
```

```
plt.tight_lawet()
plt.show()
```



3.3 Analysis of Average Training Hours by Experience

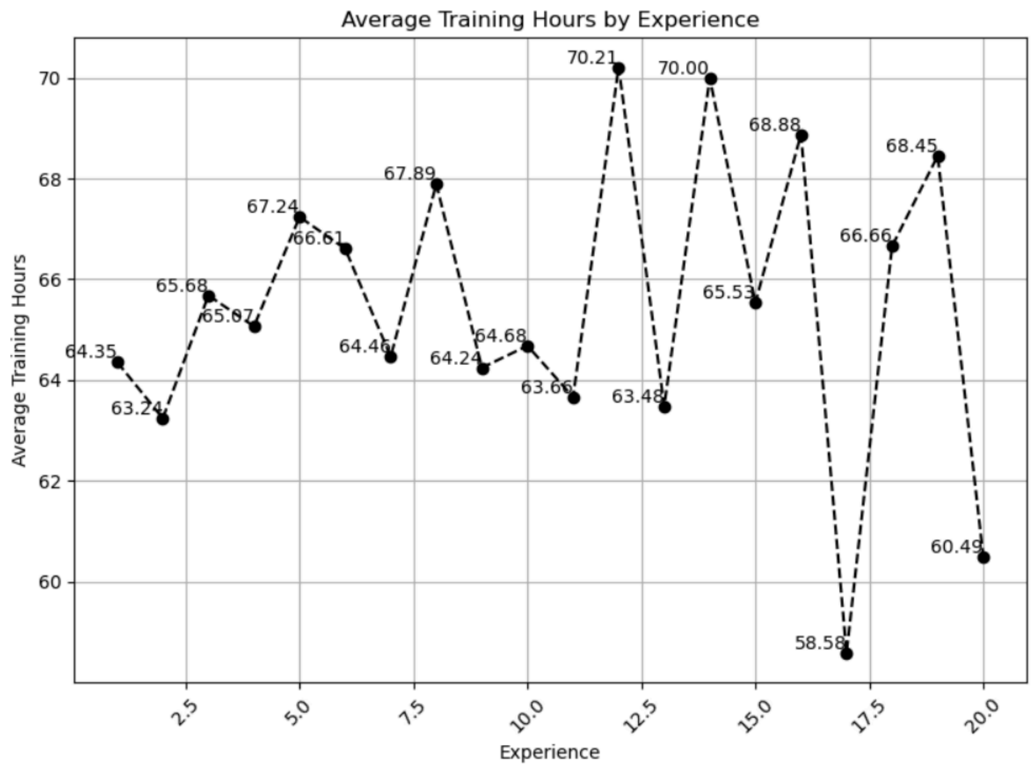
We analyzed how work experience correlates with the average training hours spent by candidates.

```
experience_training_hours = train.groupby('experience')['training_hours'].mean()
```

A line chart with markers was used to plot the average training hours against experience levels.

```
plt.figure(figsize=(8,6))
plt.plot(experience_training_hours.index, experience_training_hours.values, color='black', marker='o', linestyle='dashed')
for i, value in enumerate(experience_training_hours.values):
    plt.text(experience_training_hours.index[i], value, f'{value:.2f}', fontsize=10, ha='right', va='bottom')
plt.xlabel('Experience')
plt.ylabel('Average Training Hours')
plt.title('Average Training Hours by Experience')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_lawet()
```

plt.show()



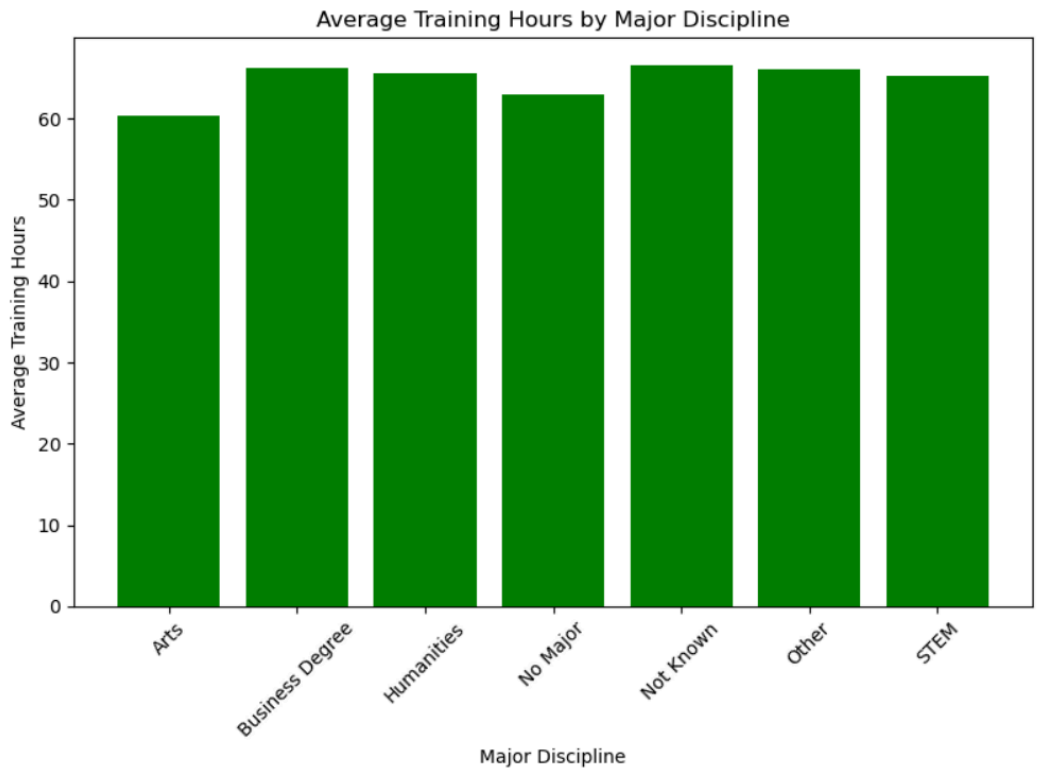
3.4 Analysis of Average Training Hours by Major Discipline

The analysis extended to the major_discipline column to see how different disciplines vary in terms of average training hours.

```
discipline_training = train.groupby('major_discipline')['training_hours'].mean()
```

A bar chart was created to represent the average training hours across different major disciplines.

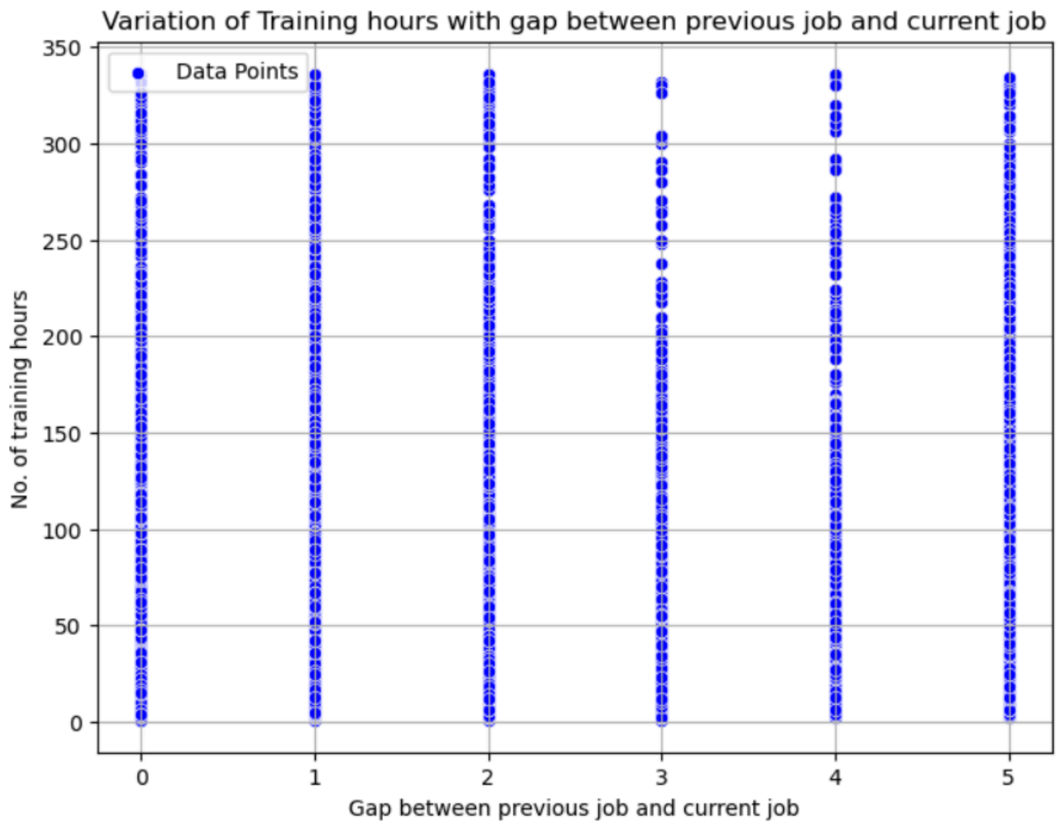
```
plt.figure(figsize=(8,6))
plt.bar(discipline_training.index, discipline_training.values, color='green')
plt.xlabel('Major Discipline')
plt.ylabel('Average Training Hours')
plt.title('Average Training Hours by Major Discipline')
plt.xticks(rotation=45)
plt.tight_lawet()
plt.show()
```



3.5 Scatter Plot of Last Job Change vs. Training Hours

To understand the relationship between the time since a candidate's last job change (last_new_job) and their training hours, we used a scatter plot.

```
plt.figure(figsize=(8,6))
sns.scatterplot(data=train, x='last_new_job', y='training_hours', color='blue', label='Data Points')
plt.xlabel('Gap between Previous Job and Current Job')
plt.ylabel('No. of Training Hours')
plt.title('Variation of Training Hours with Gap between Previous Job and Current Job')
plt.grid(True)
plt.show()
```



3.6 Numerical and Categorical Features

We separated the dataset into numerical and categorical features for further analysis.

```
num_feat = train.iloc[:, :-1].select_dtypes('number').columns
cat_feat = train.select_dtypes('object').columns
print(num_feat)
print(cat_feat)
```

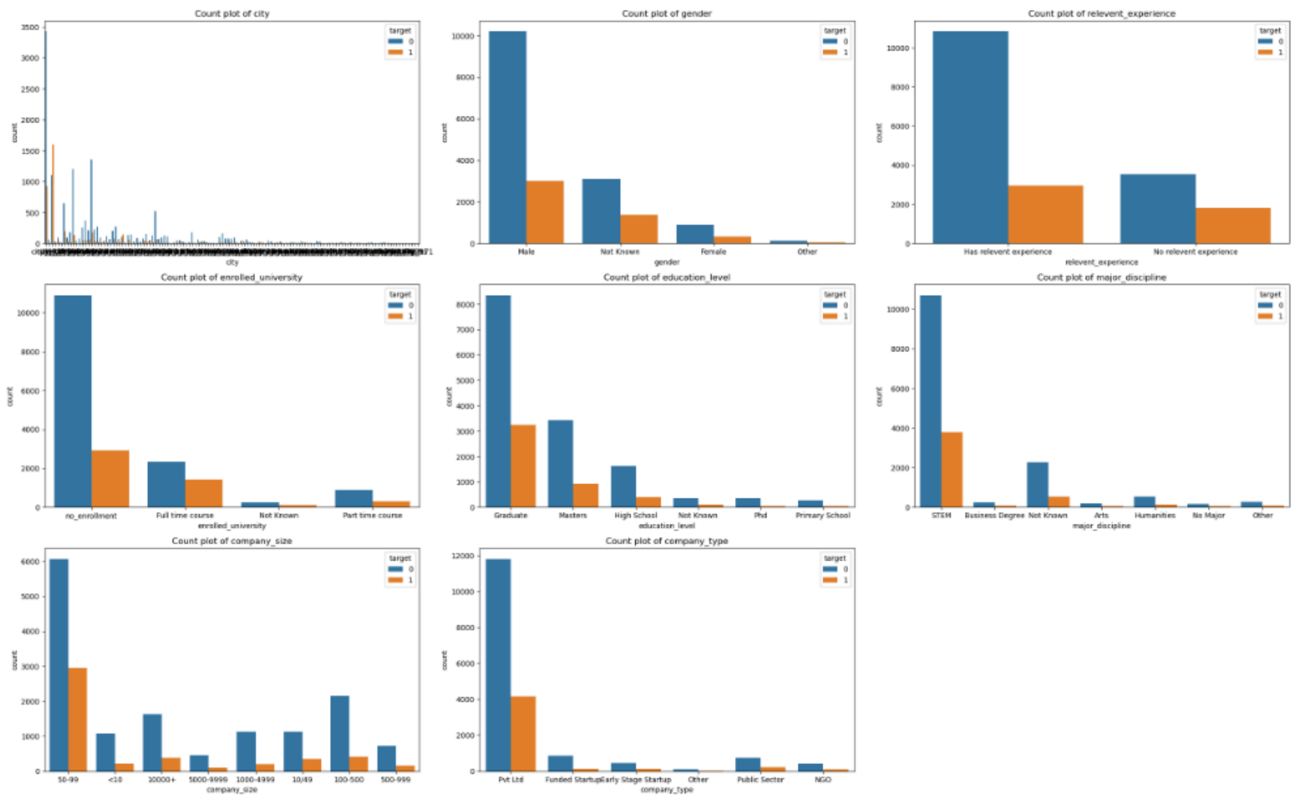
Numerical Features	Categorical Features
`enrollee_id`, `city_development_index`, `experience`, `last_new_job`, `training_hours`	`city`, `gender`, `relevent_experience`, `enrolled_university`, `education_level`, `major_discipline`, `company_size`, `company_type`

3.7 Count Plots of Categorical Features with Target Variable

To understand how different categorical features relate to the target variable, we plotted count plots for each categorical feature, segmented by the target.

```
plt.figure(figsize=(25, 20))
for i in range(len(cat_feat)):
```

```
    plt.subplot(4, 3, i + 1)
    sns.countplot(x=cat_feat[i], data=train, hue='target')
    plt.title(f'Count Plot of {cat_feat[i]}')
plt.tight_lawet()
plt.show()
```



3.8 Distribution of Education Levels

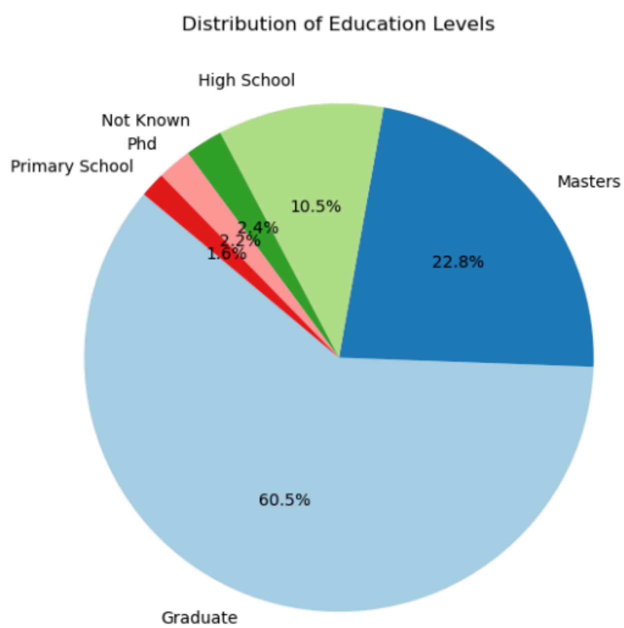
The count of occurrences for each education level was analyzed, and a pie chart was generated to visualize the distribution.

```
education_counts = train['education_level'].value_counts()
education_dict = education_counts.to_dict()
print(education_dict)
```

Education Level	Count
Graduate	11,598
Masters	4,361
High School	2,017
Not Known	460
PhD	414
Primary School	308

A pie chart displaying the distribution of education levels.

```
plt.figure(figsize=(7, 7))
plt.pie(education_counts, labels=education_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
plt.title('Distribution of Education Levels')
plt.show()
```



3.9 Distribution of Major Disciplines

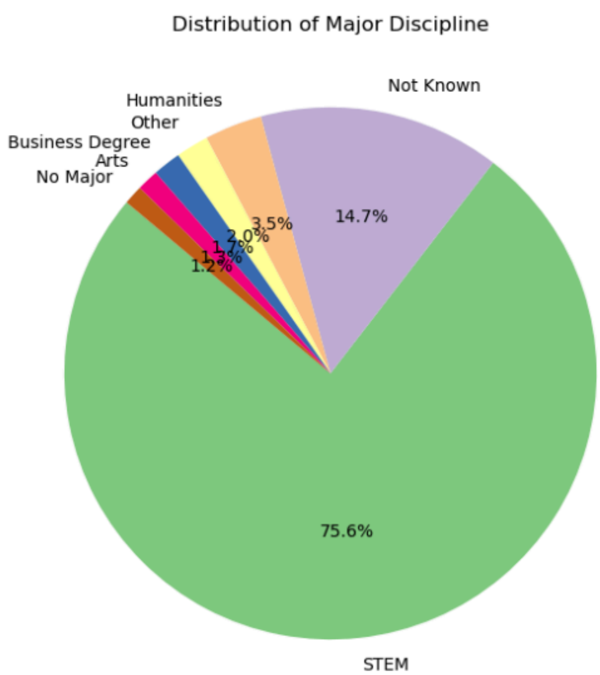
We examined the distribution of candidates' major disciplines and created a pie chart to illustrate the results.

```
major_discipline_counts = train['major_discipline'].value_counts()
major_discipline_dict = major_discipline_counts.to_dict()
print(major_discipline_dict)
```

Major Discipline	Count
STEM	14,492
Not Known	2,813
Humanities	669
Other	381
Business Degree	327
Arts	253
No Major	223

A pie chart displaying the distribution of major disciplines.

```
plt.figure(figsize=(7, 7))
plt.pie(major_discipline_counts, labels=major_discipline_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Accent.colors)
plt.title('Distribution of Major Discipline')
plt.show()
```



3.10 Distribution of Experience Levels

We analyzed the distribution of candidate experience levels and plotted a pie chart.

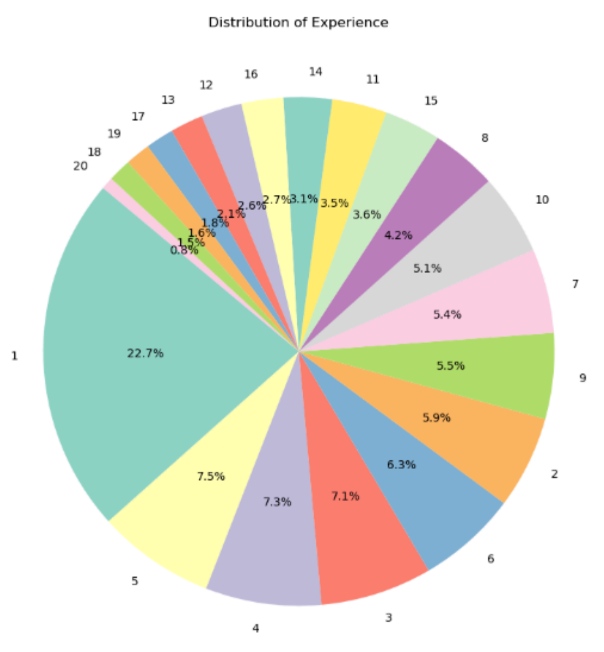
```
experience_counts = train['experience'].value_counts()
experience_dict = experience_counts.to_dict()
```

```
print(experience_dict)
```

Experience Level	Count
1	4,357
5	1,430
4	1,403
3	1,354
6	1,216
2	1,127
9	1,045
7	1,028
10	985
8	802

A pie chart displaying the distribution of experience levels.

```
plt.figure(figsize=(10, 10))
plt.pie(experience_counts, labels=experience_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Set3.colors)
plt.title('Distribution of Experience')
plt.show()
```



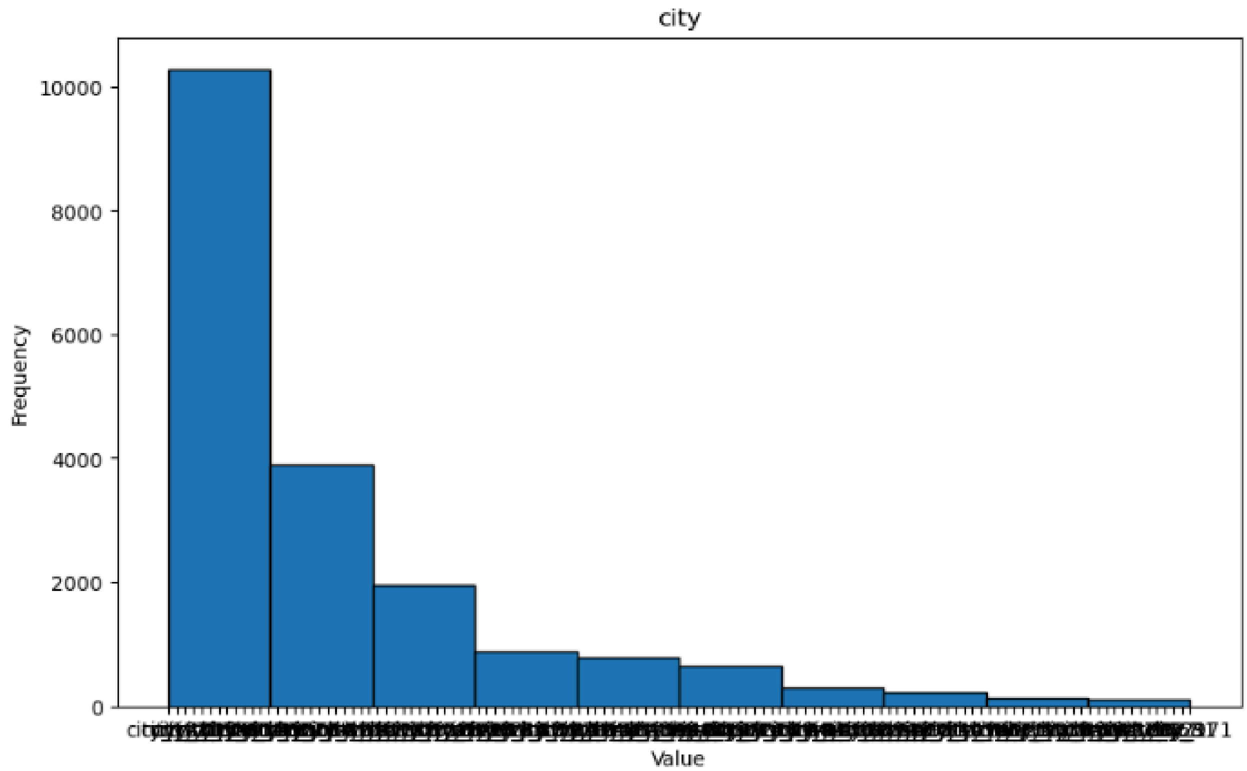
3.11 Histogram Analysis of Key Features

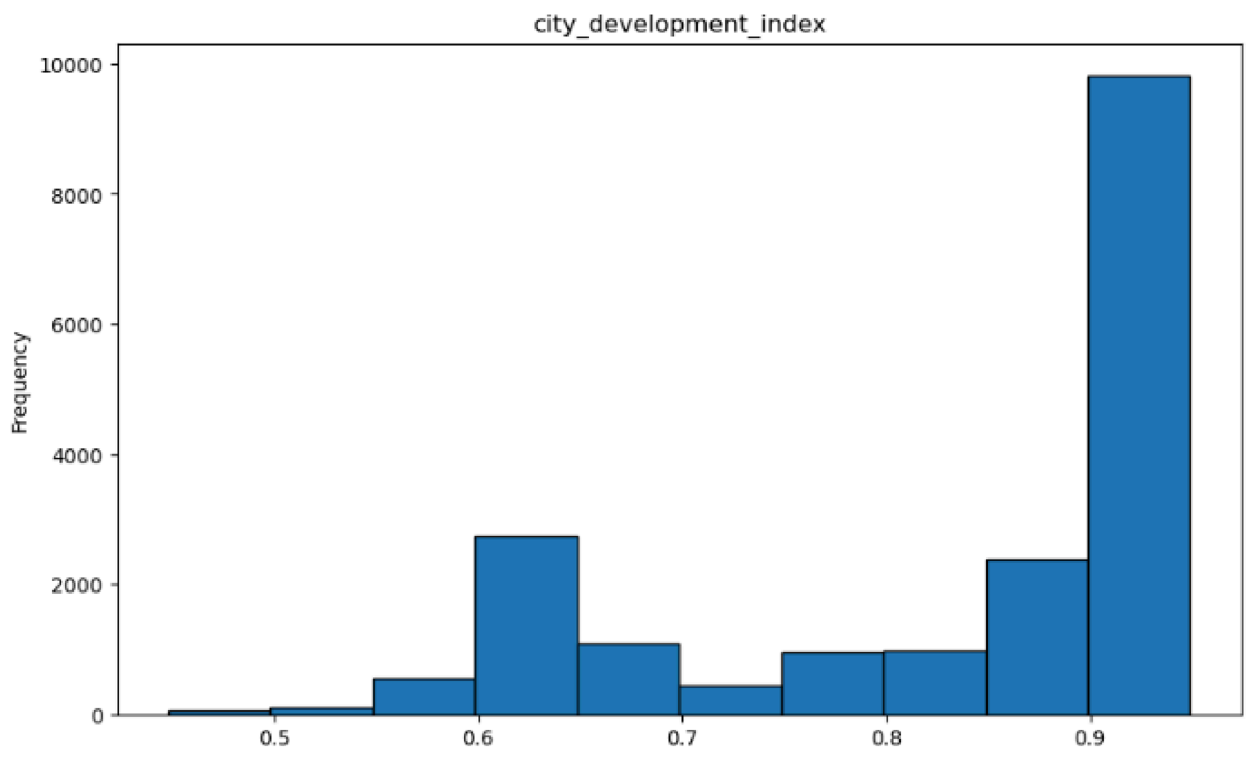
To further explore the distribution of various features in the dataset, we generated histograms for several key variables. This helps in visualizing the spread and frequency of each feature.

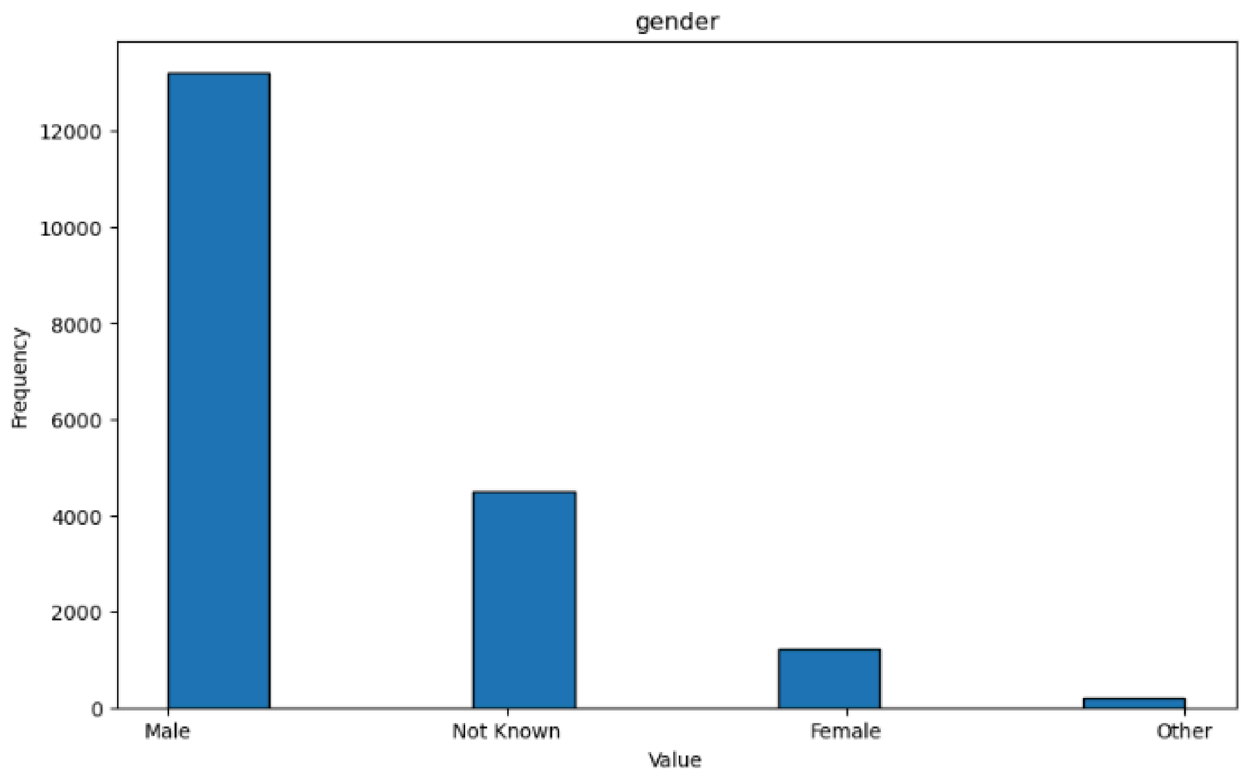
The features analyzed through histograms include:

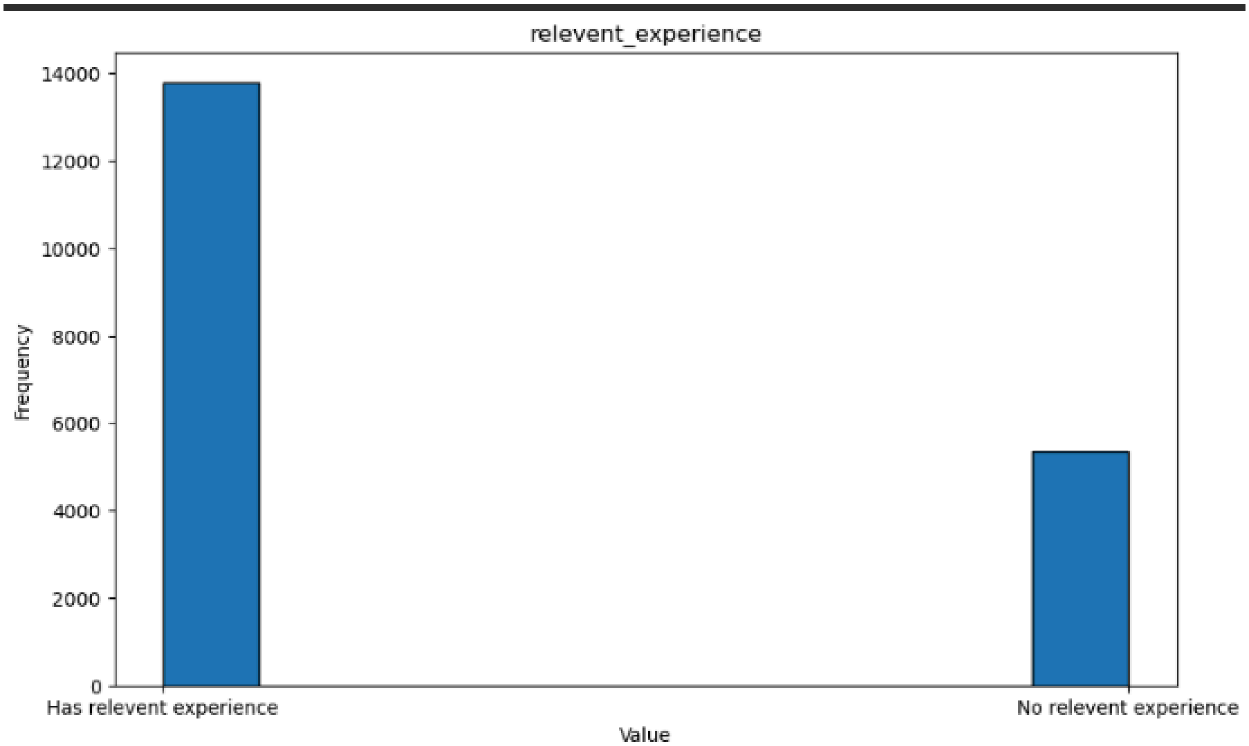
- city
- city_development_index
- gender
- relevent_experience
- enrolled_university
- education_level
- major_discipline
- experience
- company_size
- company_type
- last_new_job
- training_hours

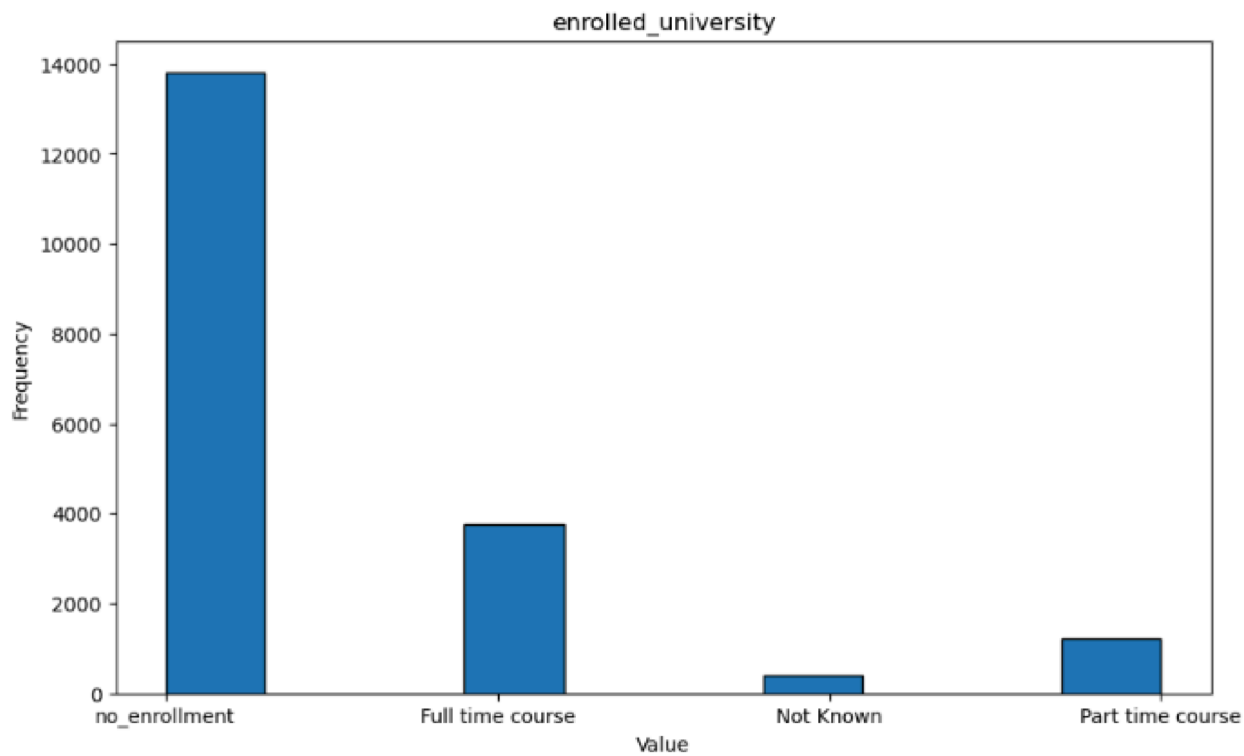
```
# List of selected columns for histogram analysis
list = ['city', 'city_development_index', 'gender', 'relevent_experience', 'enrolled_university',
'education_level', 'major_discipline', 'experience', 'company_size', 'company_type',
'last_new_job', 'training_hours']

# Loop through each feature and plot its histogram
for i in list:
   plt.figure(figsize=(10, 6))
   plt.hist(train[i], bins=10, edgecolor='black')  # Plotting histogram with 10 bins and black edges
   plt.title(i)  # Setting title as feature name
   plt.xlabel('Value')  # X-axis label
   plt.ylabel('Frequency')  # Y-axis label
   plt.show()  # Display the plot
```

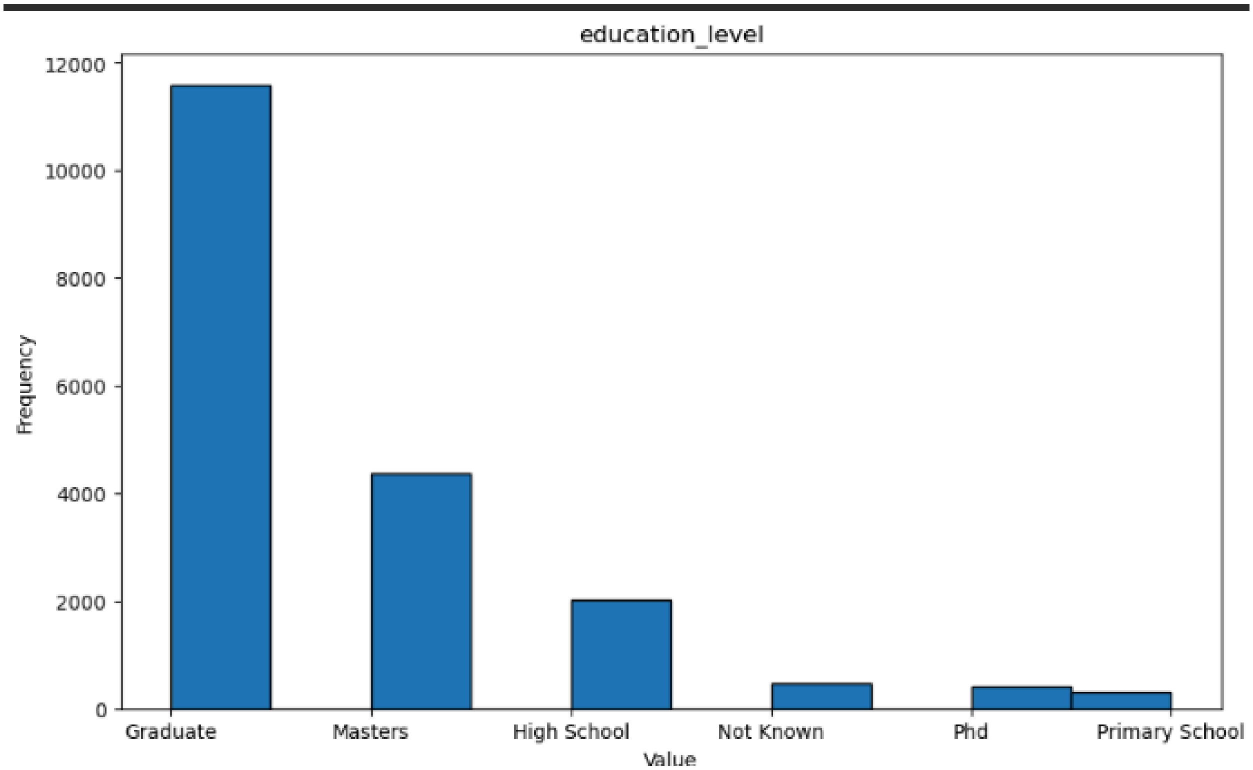

city
Frequency
Value
0
2000
4000
6000
8000
10000

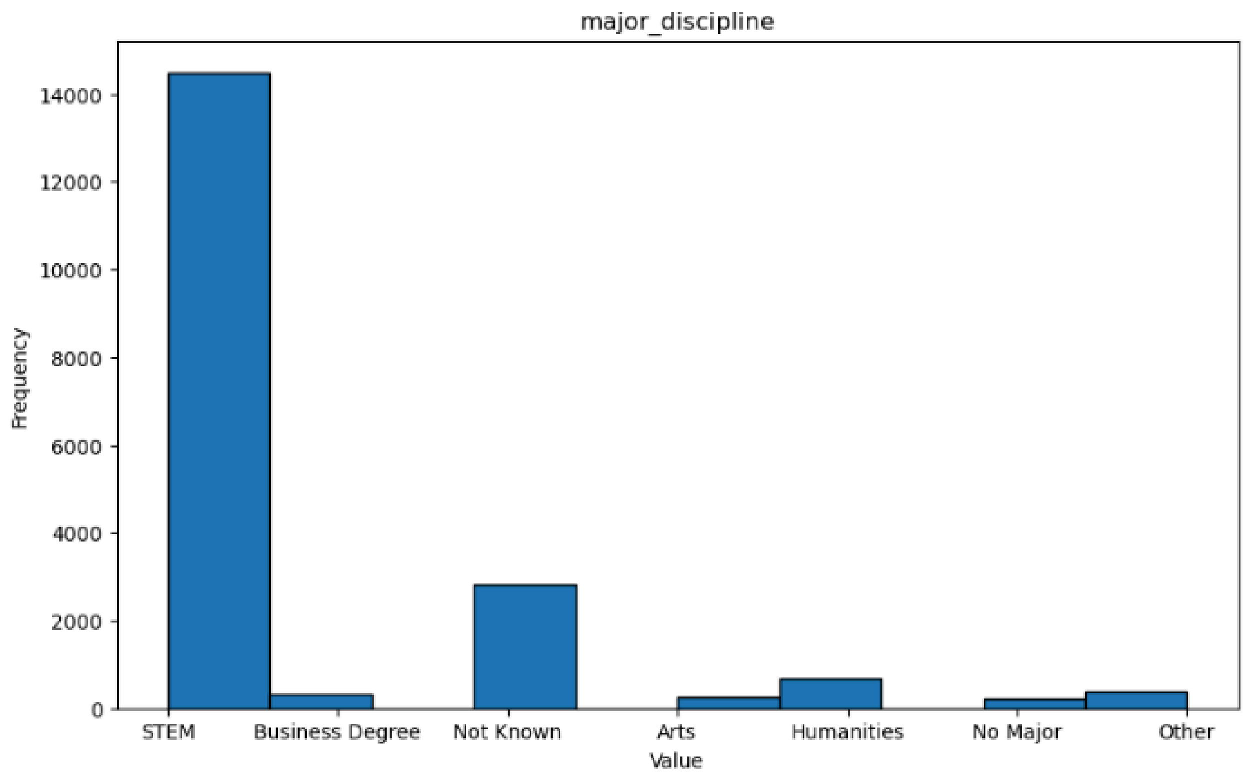

city_development_index
Frequency
0
2000
4000
6000
8000
10000
0.5
0.6
0.7
0.8
0.9

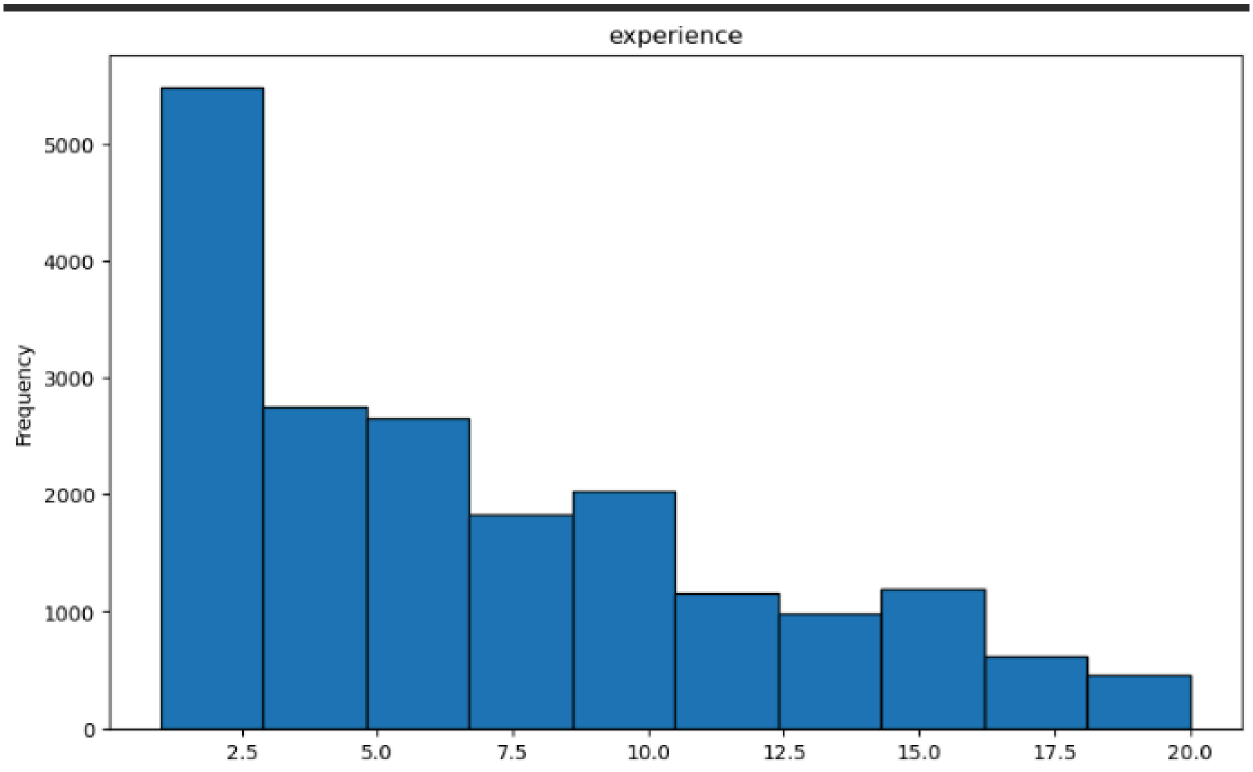

gender
Frequency
0
2000
4000
6000
8000
10000
12000
Male
Not Known
Female
Other
Value

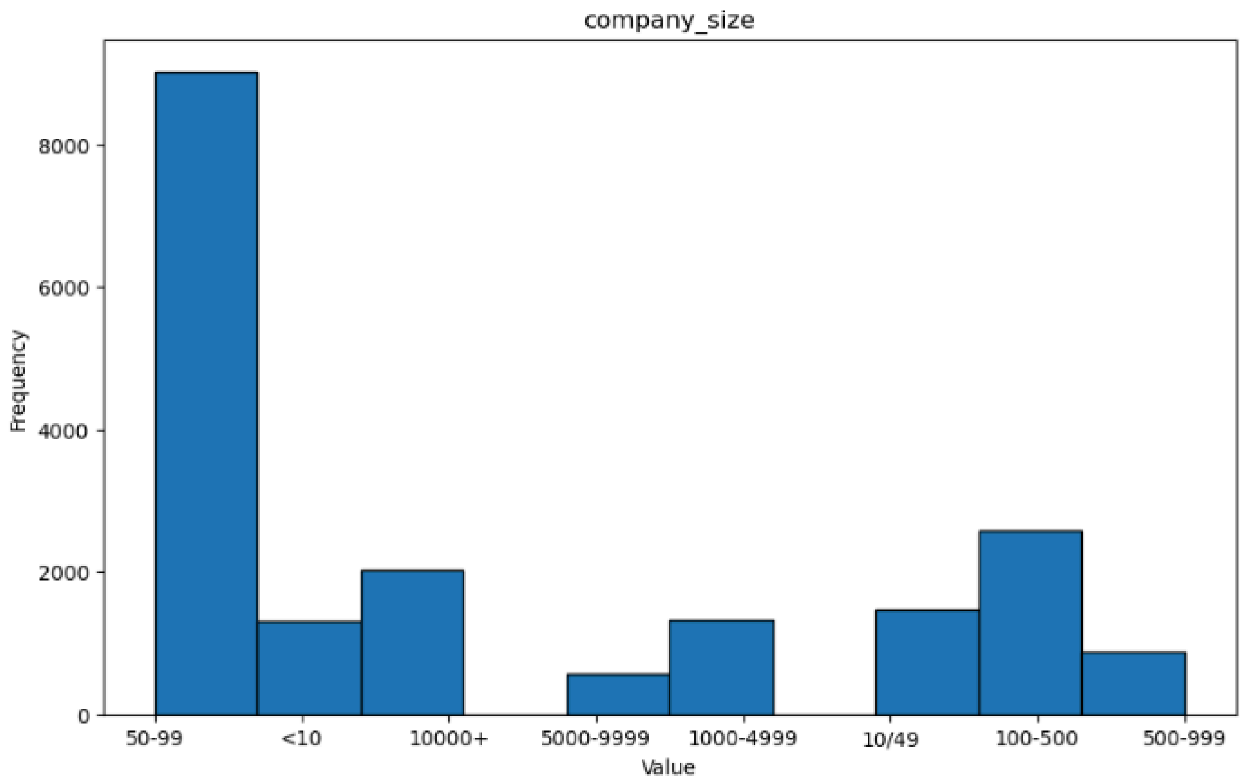

relevent_experience
Frequency
0
2000
4000
6000
8000
10000
12000
14000
Has relevent experience
No relevent experience
Value

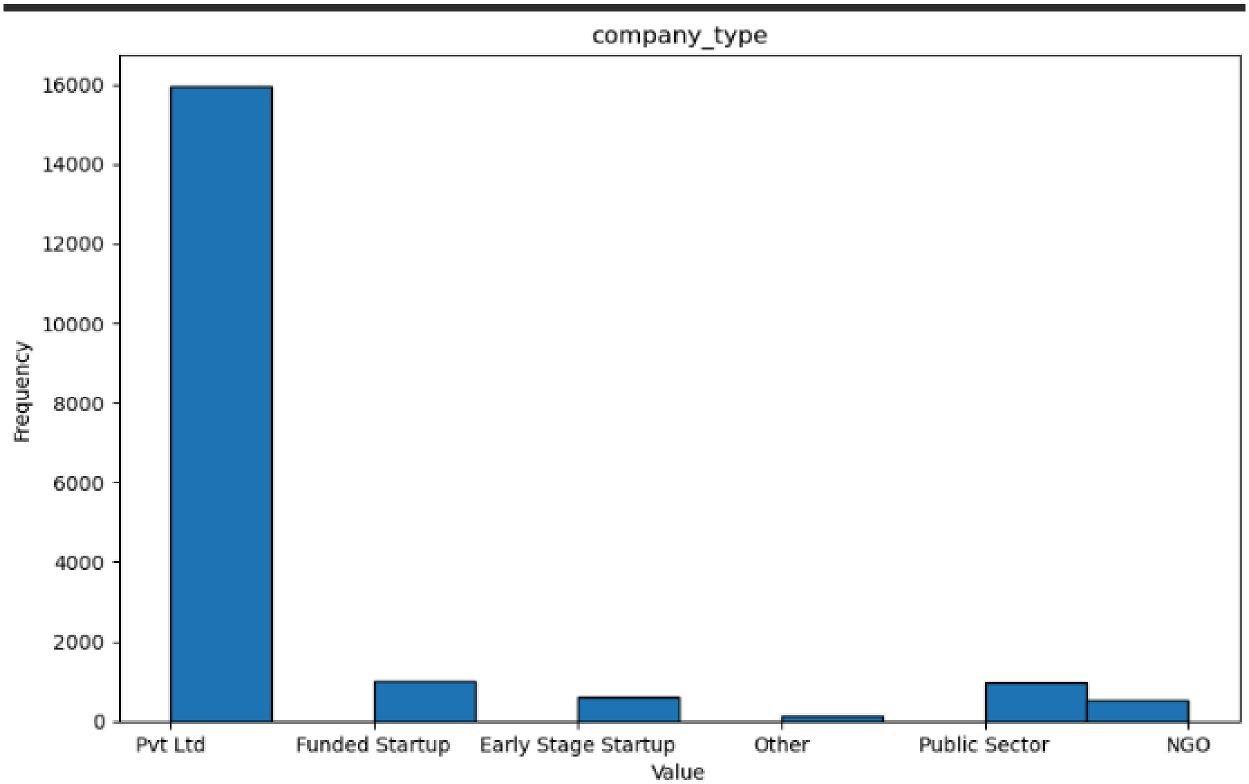

enrolled_university
14000
12000
10000
8000
6000
4000
2000
0
Frequency
no_enrollment
Full time course
Not Known
Part time course
Value

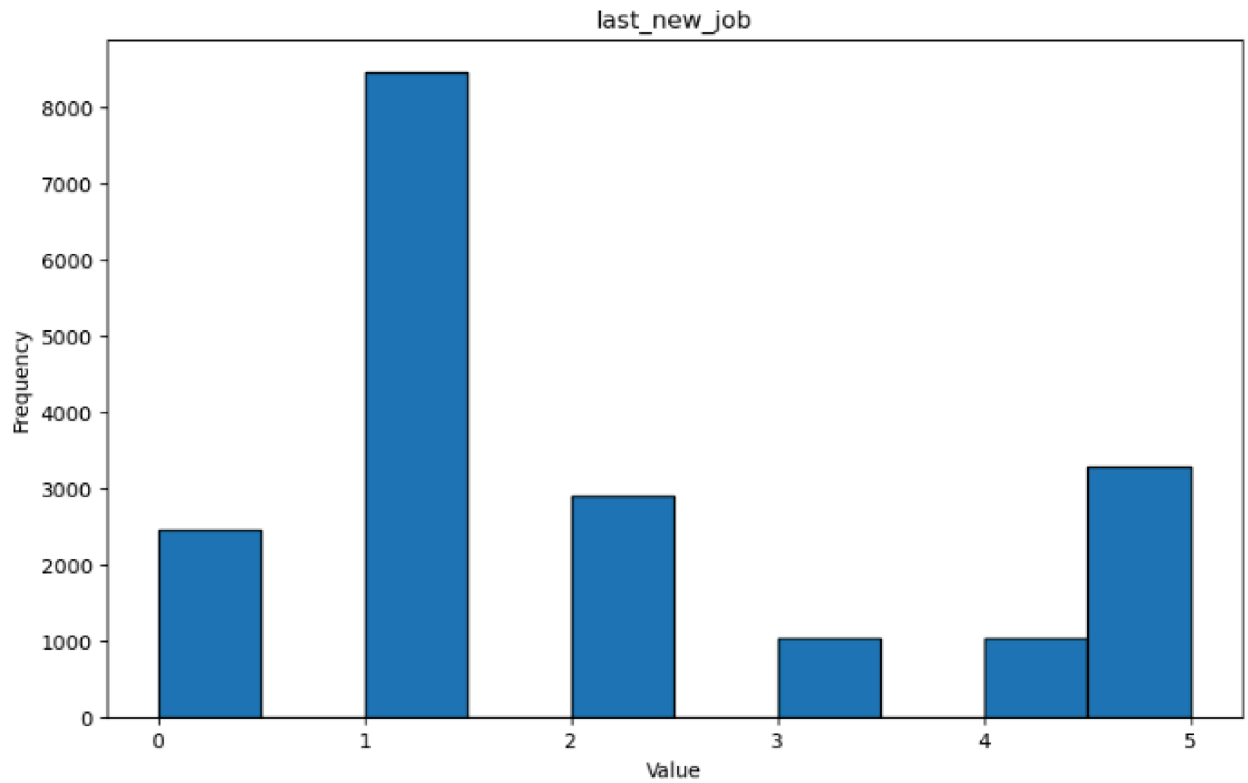

education_level
12000
10000
8000
6000
4000
2000
0
Frequency
Graduate
Masters
High School
Not Known
Phd
Primary School
Value

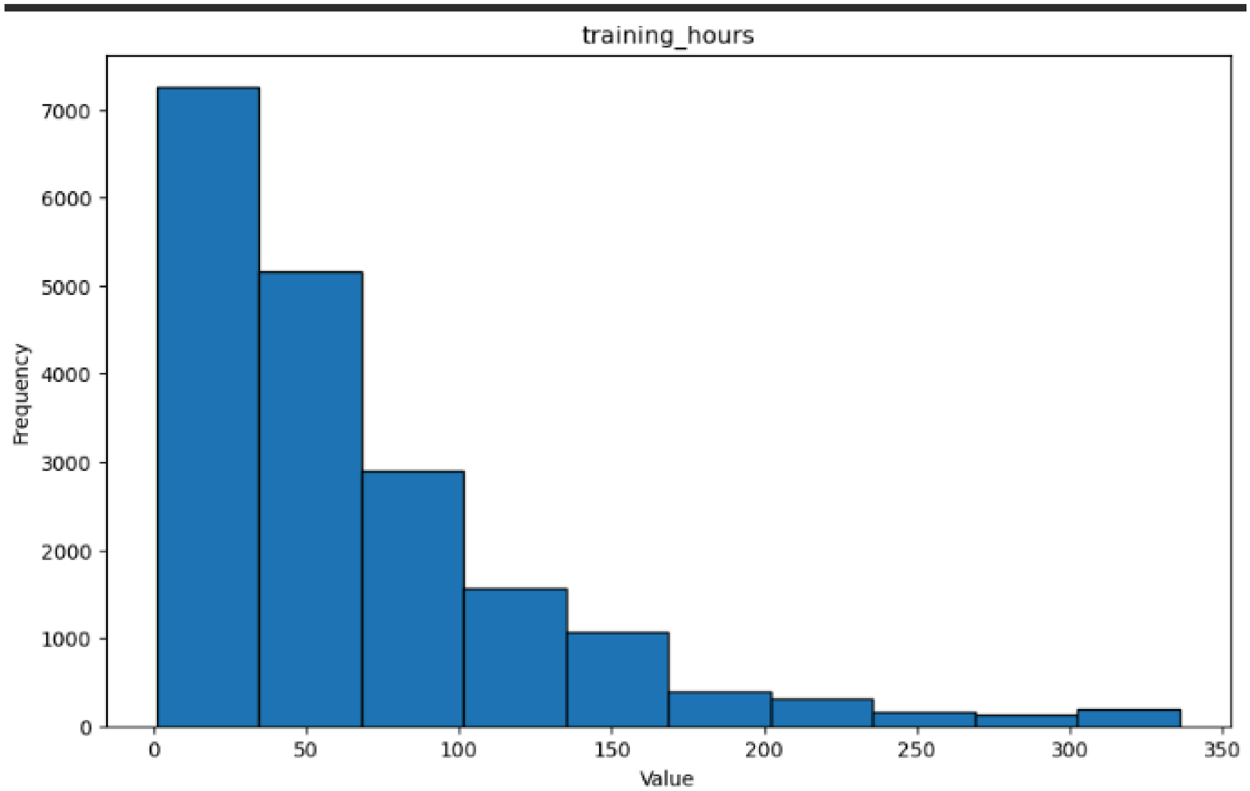

major_discipline
Frequency
14000
12000
10000
8000
6000
4000
2000
0
STEM
Business Degree
Not Known
Arts
Humanities
No Major
Other
Value


experience
Frequency
5000
4000
3000
2000
1000
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


company_size
Frequency
0
2000
4000
6000
8000
50-99
<10
10000+
5000-9999
1000-4999
10/49
100-500
500-999
Value


company_type
Frequency
0
2000
4000
6000
8000
10000
12000
14000
16000
Pvt Ltd
Funded Startup
Early Stage Startup
Other
Public Sector
NGO
Value


last_new_job
Frequency
0
1000
2000
3000
4000
5000
6000
7000
8000
0
1
2
3
4
5
Value


training_hours
Frequency
0
1000
2000
3000
4000
5000
6000
7000
0
50
100
150
200
250
300
350
Value

Insights:

- **City and City Development Index**: The histograms for these features can provide insights into the geographic distribution of candidates and their city's development.
- **Gender and Relevant Experience**: These plots help in understanding the gender distribution and the ratio of candidates with prior relevant experience.
- **Enrolled University, Education Level, and Major Discipline**: These variables reflect the academic background of the candidates, showing how many are still studying, their highest qualifications, and the fields they specialize in.
- **Experience and Company Information**: The histograms for experience, company_size, and company_type provide a view of candidates' work history and the types of companies they have worked for.
- **Last New Job and Training Hours**: These histograms can shed light on how long candidates have been between jobs and how many hours of training they've completed.

The histograms provide a clearer picture of how different factors vary across the dataset, revealing key patterns that can guide further analysis.

4. Encoding, Scaling, and Normalization

In this section, we transform categorical and numerical variables to make the data suitable for machine learning models. The process involves **encoding** categorical features, **scaling** numerical variables, and **normalizing** the data.

4.1. Encoding

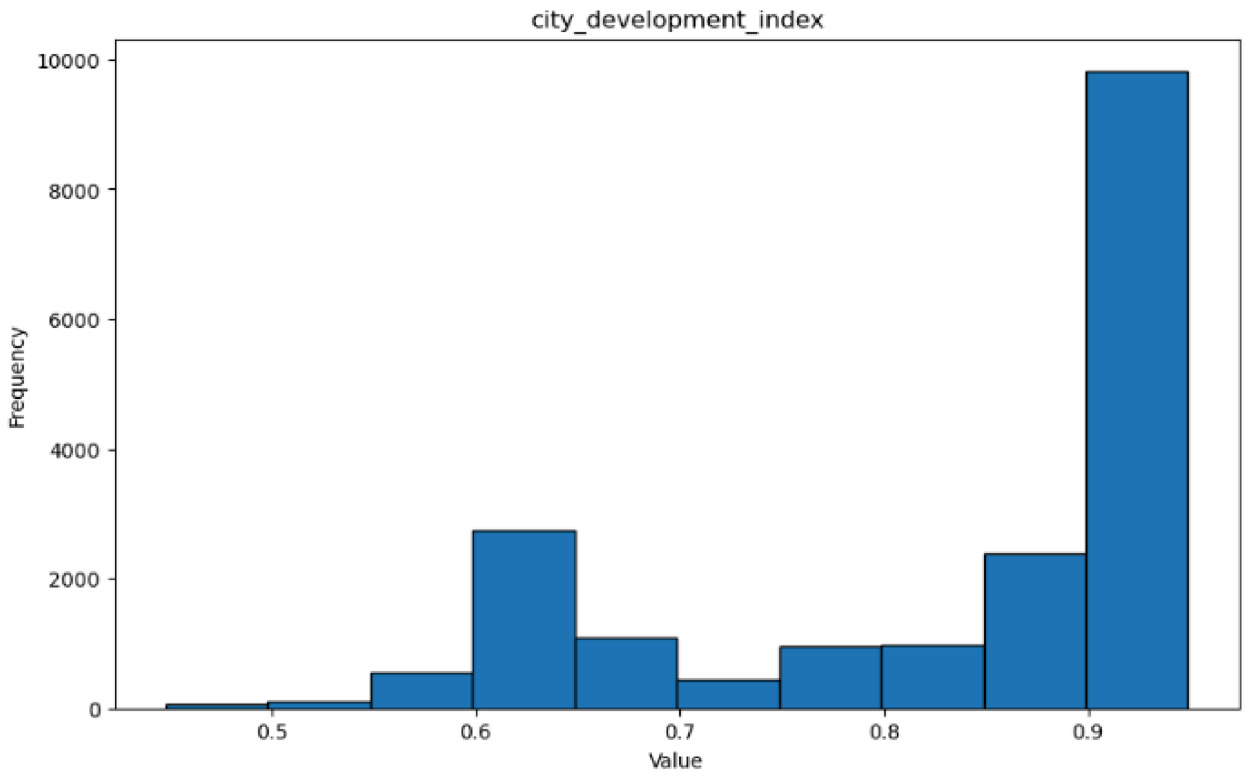
Encoding is essential to convert categorical data into numerical values. Below are the steps followed:

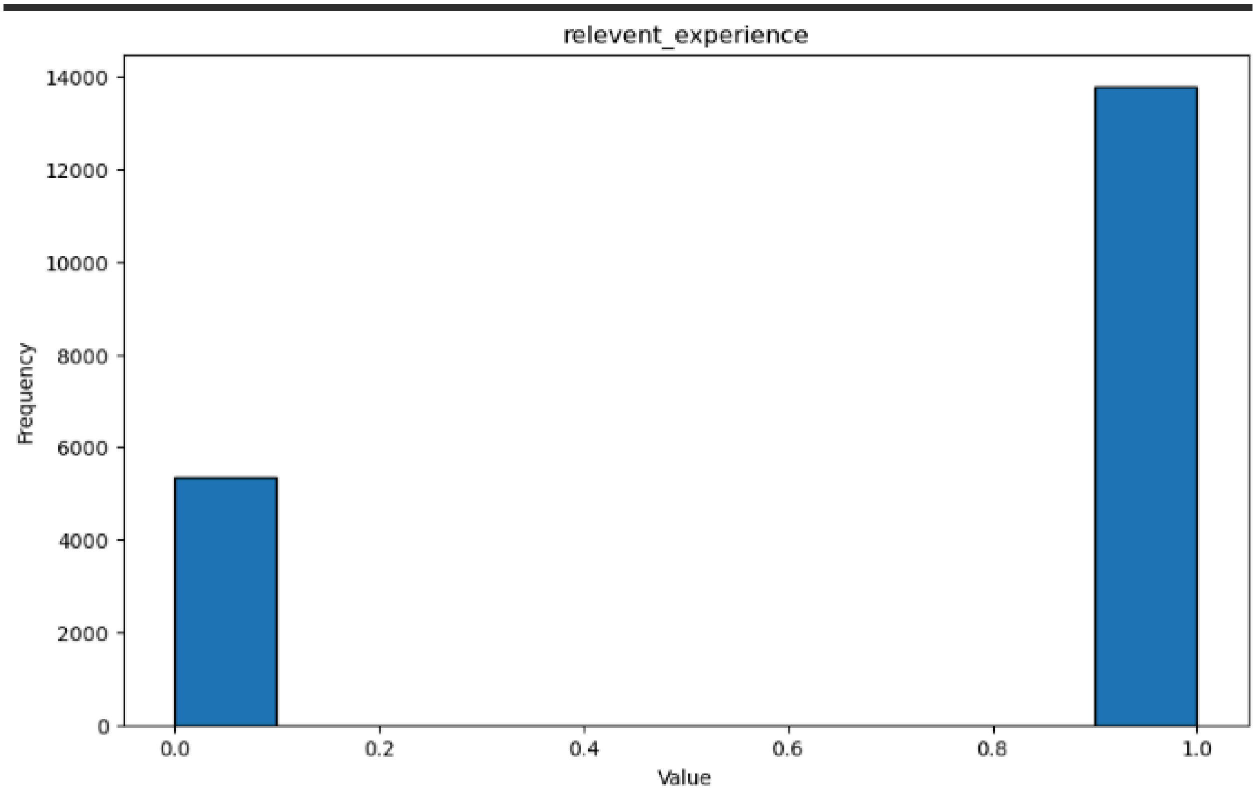
- **Label Encoding** is applied to the city column, as it has many unique values.
- **One-Hot Encoding** is used for columns like gender, enrolled_university, education_level, major_discipline, company_size, and company_type.

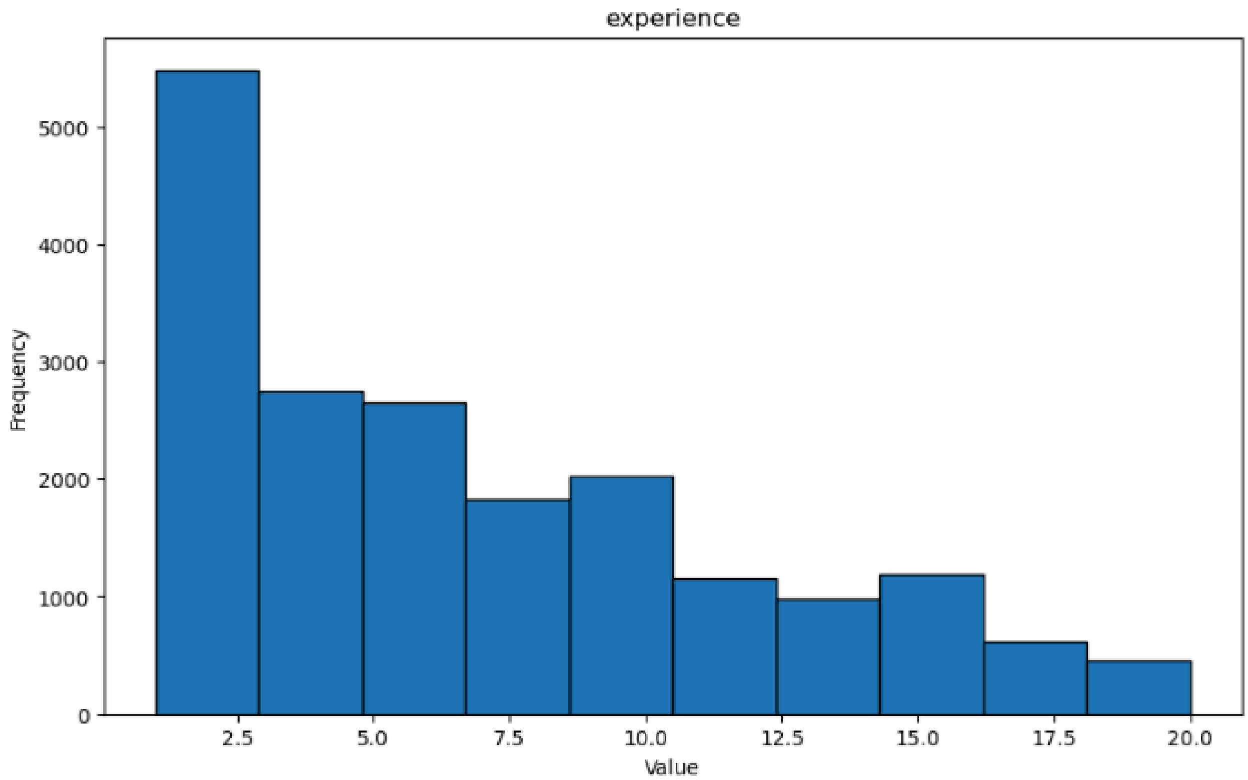
```
# Label encoding for the 'city' column

label_encoder = LabelEncoder()

train['city_encoded'] = label_encoder.fit_transform(train['city'])

test['city_encoded'] = label_encoder.fit_transform(test['city'])
```

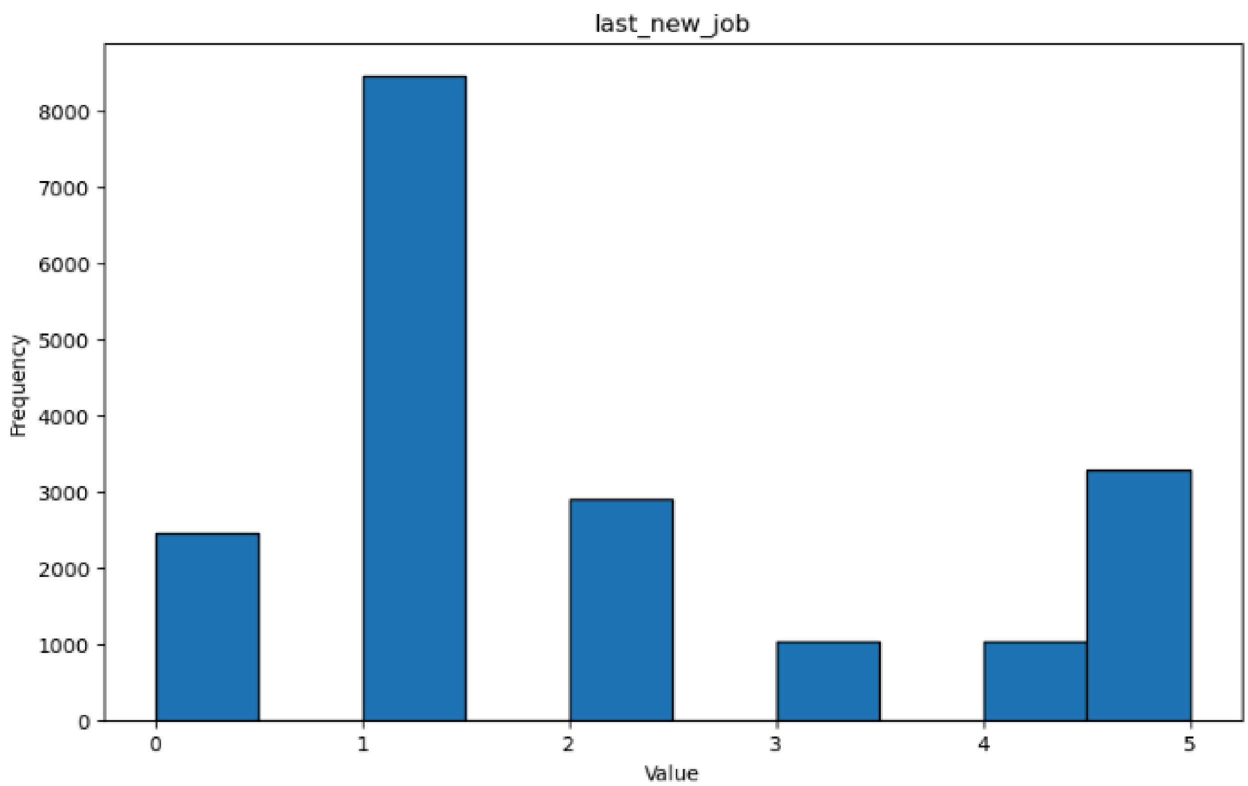
```
# One-hot encoding for 'gender'

encoded_train = pd.get_dummies(train, columns=['gender'])

encoded_test = pd.get_dummies(test, columns=['gender'])


# Converting 'relevent_experience' to binary values

encoded_train['relevent_experience'] = encoded_train['relevent_experience'].replace('Has relevent experience', 1)

encoded_train['relevent_experience'] = encoded_train['relevent_experience'].replace('No relevent experience', 0)

encoded_test['relevent_experience'] = encoded_test['relevent_experience'].replace('Has relevent experience', 1)

encoded_test['relevent_experience'] = encoded_test['relevent_experience'].replace('No relevent experience', 0)


# One-hot encoding for other categorical columns

encoded_train = pd.get_dummies(encoded_train, columns=['enrolled_university', 'education_level', 'major_discipline', 'company_size', 'company_type'])

encoded_test = pd.get_dummies(encoded_test, columns=['enrolled_university', 'education_level', 'major_discipline', 'company_size', 'company_type'])


# Plotting histogram

for i in list1:

    plt.figure(figsize=(10, 6))

    plt.hist(encoded_train[i], bins=10, edgecolor='black')

    plt.title(i)

    plt.xlabel('Value')
```

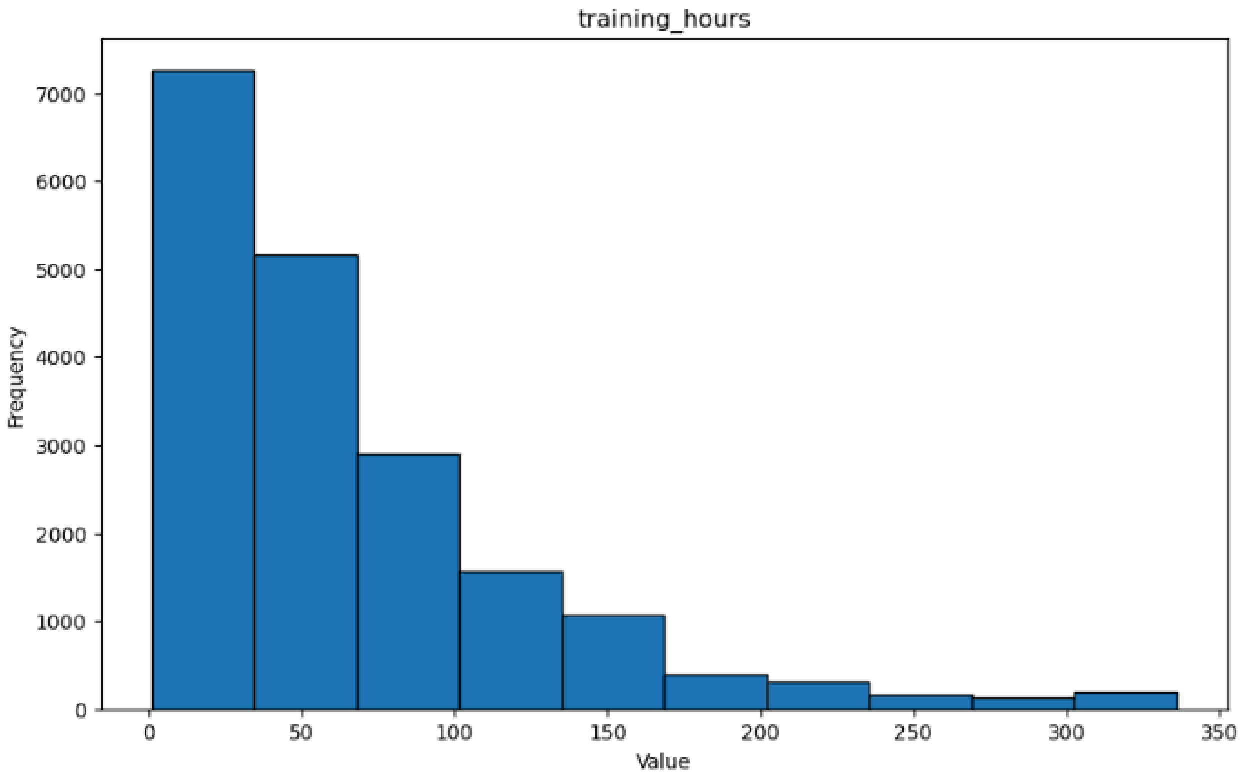
```
plt.ylabel('Frequency')

plt.show()
```

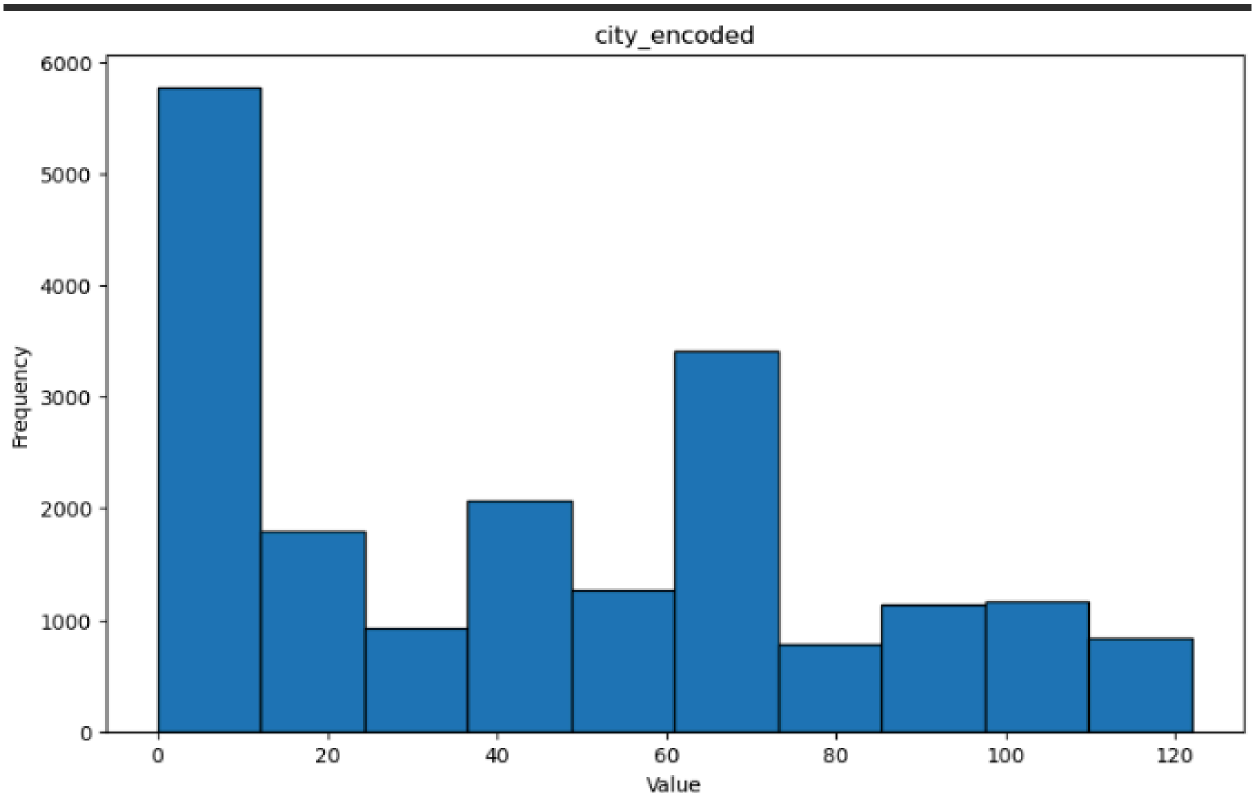

city_development_index
Frequency
Value
0
2000
4000
6000
8000
10000
0.5
0.6
0.7
0.8
0.9


relevent_experience
Frequency
Value
0
2000
4000
6000
8000
10000
12000
14000
0.0
0.2
0.4
0.6
0.8
1.0


experience
Frequency
Value
0
1000
2000
3000
4000
5000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


last_new_job
Frequency
Value
0
1000
2000
3000
4000
5000
6000
7000
8000
0
1
2
3
4
5


training_hours
Frequency
0
1000
2000
3000
4000
5000
6000
7000
0
50
100
150
200
250
300
350
Value


city_encoded
Frequency
0
1000
2000
3000
4000
5000
6000
0
20
40
60
80
100
120
Value

4.2. Scaling

Scaling is important to bring all features to the same range, ensuring that one feature doesn't dominate others. We use **Quantile Transformation** to scale numerical variables.

```
# List of columns to scale using Quantile Transformation

list2 = ['city_development_index', 'experience', 'last_new_job', 'training_hours', 'city_encoded']


# Apply Quantile Transformation to the selected columns

for i in list2:

    qt = QuantileTransformer(output_distribution='normal')

    encoded_train[i] = qt.fit_transform(encoded_train[[i]])

    encoded_test[i] = qt.fit_transform(encoded_test[[i]])
```

4.3. Normalization

MinMaxScaler is used to normalize the data. This ensures that values lie between 0 and 1, making it easier for models to converge.

python

Copy code

```
# Initialize the MinMaxScaler

scaler = MinMaxScaler()

# Apply MinMax scaling to selected columns

columns_to_scale = ['city_development_index', 'experience', 'last_new_job', 'training_hours', 'city_encoded']

for column in columns_to_scale:

    encoded_train[[column]] = scaler.fit_transform(encoded_train[[column]])

    encoded_test[[column]] = scaler.fit_transform(encoded_test[[column]])
```

Preview of Encoded and Scaled Data:

```
encoded_train.head()
```

This final step ensures that all categorical variables are encoded, numerical variables are scaled and normalized, and the dataset is ready for model training.

5. Deleting Unnecessary Columns

To remove unnecessary columns that do not contribute to model training, we drop the enrollee_id and city columns from both the training and testing datasets.

```
# Deleting 'enrollee_id' and 'city' columns from both encoded_train and encoded_test

encoded_train = encoded_train.drop(['enrollee_id', 'city'], axis=1)

encoded_test = encoded_test.drop(['enrollee_id', 'city'], axis=1)

# Preview the dataset after dropping the columns

encoded_train.head()
```

Next, we define the feature set X and the target variable y.

```
# Defining the feature matrix X (all columns except 'target')

X = encoded_train.drop(["target"], axis=1)

# Defining the target variable y (the 'target' column)

y = encoded_train['target']
```

6. Train-Test Split

We split the dataset into training and testing sets using train_test_split. This ensures that 30% of the data is reserved for testing, while 70% is used for training the model.

```
# Splitting the dataset into training and testing sets (70% train, 30% test)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

This step is essential to evaluate how well the model generalizes to unseen data. The training set is used to train the model, and the testing set is used to assess its performance.

7. Modeling

7.1. Selecting an Appropriate Model

Here, we used **LazyClassifier** to quickly try out multiple classification models and compare their performance. It helped we get a baseline understanding of which models might work best.

```
# Fitting the LazyClassifier on the training and testing data

clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)

models, predictions = clf.fit(X_train, X_test, y_train, y_test)

models  # Output shows the performance of different models
```

7.2. Fitting a Model (AdaBoost Classifier)

Next, we chose to use an **AdaBoost Classifier** with a weak decision tree learner as the base estimator.

```
# Training the AdaBoost classifier with a decision tree as the base estimator

base_estimator = DecisionTreeClassifier(max_depth=1)  # Weak learner

model = AdaBoostClassifier(n_estimators=50, random_state=42)

model.fit(X_train, y_train)

# Making predictions on the test set

y_pred = model.predict(X_test)

# Evaluating the model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy Score: {accuracy:.2f}')
```

This model gave an accuracy score of **77%**.

7.3. Performing Cross-Validation

To ensure the model's robustness, we performed 5-fold cross-validation on the training data.

```
# Performing 5-fold cross-validation
cv_scores = cross_val_score(model, X_train, y_train, cv=5)
# Calculating the mean accuracy across the 5 folds
mean_cv_accuracy = cv_scores.mean()
print(f"Mean Cross-Validation Accuracy: {mean_cv_accuracy:.2f}")
```

This resulted in a mean cross-validation accuracy of **78%**.

7.4. Hyperparameter Tuning

We used **GridSearchCV** to find the optimal hyperparameters for the AdaBoost model.

```
# Defining the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1, 10]
}
# Running Grid Search
grid_search = GridSearchCV(AdaBoostClassifier(), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)
# Evaluating the best model
best_model = grid_search.best_estimator_
```

```
y_pred = best_model.predict(X_test)

best_params = grid_search.best_params_

print("Best Hyperparameters:", best_params)

# Accuracy score of the best model

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy Score: {accuracy:.2f}')
```

The best hyperparameters were:

- **learning_rate: 0.01**
- **n_estimators: 50**

The tuned model achieved an accuracy score of **0.78**.

7.5. Fitting the Data with Best Parameters

We refit the AdaBoost model using the best hyperparameters and evaluated it again.

```
# Refit the AdaBoost model with the best parameters

final_model = AdaBoostClassifier(**best_params)

final_model.fit(X_train, y_train)


# Making predictions

y_pred = final_model.predict(X_test)

# Accuracy and classification report

accuracy = accuracy_score(y_test, y_pred)

print(f"Test Accuracy: {accuracy:.2f}")

print("Classification Report:")

print(classification_report(y_test, y_pred))
```

```
print("Confusion Matrix:")

print(confusion_matrix(y_test, y_pred))
```

The test accuracy was **0.78**, and the classification report showed:

- **Precision** and **recall** for class 0 (negative class) were higher than for class 1 (positive class).
- The confusion matrix showed that more false negatives were predicted (i.e., actual class 1 predicted as 0).

7.6. Performing Cross-Validation

We performed cross-validation on the final model to validate its performance further.

```
# Cross-validation on the final model

cv_scores = cross_val_score(final_model, X_train, y_train, cv=5, scoring='accuracy')

print(f"Cross-Validated Accuracy: {cv_scores.mean():.2f}")
```

The cross-validated accuracy was **0.78**.

7.7. Trying the Model on Unseen Data

Finally, we made predictions on the test dataset (unseen data) using the final trained model.

```
# Making predictions on unseen data (encoded_test)

y_unseen_pred = final_model.predict(encoded_test)

# Creating a DataFrame to store the predictions

encoded_test['pred_target'] = y_unseen_pred
```

This completes the modeling process, with predictions stored in the encoded_test DataFrame.