Mini-project report
Applied Statistics and Experimental Design
Topic: Missing data handling

Group 18
Tran Cong Duy - 20214885
Tran Hoang Duong - 20214888
Nguyen Hai Duong - 20214887
Nhu Minh Ha - 20212784
Nguyen Trung Truc - 20214936

Table of content:

1. Introduction

The present report aims to explore and analyze a dataset containing information about job applicants, referred to as "enrollees." The dataset encompasses various attributes that provide valuable insights into the background and characteristics of these individuals.

The dataset contains valuable information for various data analysis tasks, including predictive modeling, classification, and pattern recognition. However, like many real-world datasets, this dataset may contain missing values, which can adversely impact the quality and reliability of data analysis results.

To ensure the robustness of our data analysis and modeling efforts, we will address the issue of missing data by employing appropriate data imputation techniques. In this report, we will focus on exploring the "CCA Imputation" method, which leverages Canonical Correlation Analysis to estimate missing values based on correlations between variables.

The report will describe the steps involved in differences in the Imputation process, its underlying principles, and its application to the dataset at hand.

Through this analysis, we seek to gain valuable insights into the characteristics of job applicants and identify key factors that contribute to their success in the application process. These insights can be invaluable for human resources departments, recruitment agencies, and organizations seeking to optimize their hiring and talent acquisition strategies.

2. Data understanding

The dataset includes the following columns:

enrollee_id: A unique identifier for each job applicant.
city: The city where the applicant is based.
city_development_index: A measure indicating the city's level of development, ranging from 0 to 1.
gender: The gender of the applicant (Male, Female, or unspecified).
relevent_experience: Indicates whether the applicant possesses relevant work experience or not.

enrolled_university: The type of university enrollment of the applicant (e.g., Full-time, Part-time, or no enrollment).
education_level: The highest level of education attained by the applicant.
major_discipline: The major discipline or field of study.
experience: The number of years of work experience.
company_size: The size of the company where the applicant is currently employed.
company_type: The type of company where the applicant is currently employed (e.g., Private Limited, Funded Startup, etc.).
training_hours: The number of hours spent on training.
target: A binary target variable indicating whether the applicant was successful (1) or unsuccessful (0) in the application process.

	enrollee_id	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	major_discipline	experience	company_size	company_type	training_hours	target
0	8949	city_103	0.920	Male	Has relevent experience	no_enrollment	Graduate	STEM	20.0	NaN	NaN	36.0	1.0
1	29725	city_40	0.776	Male	No relevent experience	no_enrollment	Graduate	STEM	15.0	50-99	Pvt Ltd	47.0	0.0
2	11561	city_21	0.624	NaN	No relevent experience	Full time course	Graduate	STEM	5.0	NaN	NaN	83.0	0.0
3	33241	city_115	0.789	NaN	No relevent experience	NaN	Graduate	Business Degree	0.0	NaN	Pvt Ltd	52.0	1.0
4	666	city_162	0.767	Male	Has relevent experience	no_enrollment	Masters	STEM	20.0	50-99	Funded Startup	8.0	0.0

Figure 1: Example data set

3. Missing data handling methods

3.1. Complete case analysis (CCA)

3.1.1. CCA Imputation

This section presents the CCA Imputation method, which was employed as one of the techniques to handle missing data in the dataset.

3.1.2. Overview of CCA Imputation

The primary goal of CCA Imputation is to estimate the missing values in the dataset by leveraging the correlations between the variables with missing data and the other relevant variables in the dataset. By utilizing the relationships between these variables, the CCA Imputation method attempts to provide accurate and informed imputations while preserving the underlying structure and patterns in the data.

3.1.3. Methodology

The CCA Imputation process involves several key steps:
Identify Variables with Missing Values: In the initial step, we identify the variables within the dataset that contain missing values. These variables constitute the target set for the CCA Imputation.
Estimate Missing Values: Using the complete dataset after removing rows containing NaN values, we can visually compare it with the original dataset through visualization

3.1.4. Evaluation and Considerations

While the CCA Imputation method can provide valuable imputations, its effectiveness heavily relies on the appropriate selection of predictor variables. Careful consideration must be given to the choice of predictor variables to ensure meaningful and accurate imputations.
Furthermore, it is essential to assess the quality and reliability of the imputed values. This evaluation can be conducted by comparing the imputed values with the original data (if available) or by analyzing the imputed data in combination with other analyses or modeling tasks.

3.1.5. Application in Our Study

In our data science project, we applied the CCA Imputation method to address the issue of missing data in the dataset. We utilized the relationships between variables to impute missing values and enhance the completeness of our dataset.

First, we have the overview of dataset:

	enrollee_id	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	major_discipline	experience	company_size	company_type	training_hours	target
0	8949	city_103	0.920	Male	Has relevent experience	no_enrollment	Graduate	STEM	20.0	NaN	NaN	36.0	1.0
1	29725	city_40	0.776	Male	No relevent experience	no_enrollment	Graduate	STEM	15.0	50-99	Pvt Ltd	47.0	0.0
2	11561	city_21	0.624	NaN	No relevent experience	Full time course	Graduate	STEM	5.0	NaN	NaN	83.0	0.0
3	33241	city_115	0.789	NaN	No relevent experience	NaN	Graduate	Business Degree	0.0	NaN	Pvt Ltd	52.0	1.0
4	666	city_162	0.767	Male	Has relevent experience	no_enrollment	Masters	STEM	20.0	50-99	Funded Startup	8.0	0.0

Next, we calculating the percentage of missing values in each column of the DataFrame 'df':

```
df.isnull().mean()*100
#tính tỷ lệ phần trăm giá trị thiếu (missing values) trong mỗi cột của DataFrame 'df'
```

```
enrollee_id              0.000000
city                     0.000000
city_development_index   2.500261
gender                  23.530640
relevent_experience      0.000000
enrolled_university      2.014824
education_level          2.401086
major_discipline        14.683161
experience               0.339284
company_size            30.994885
company_type            32.049274
training_hours           3.998330
target                   0.000000
dtype: float64
```

Our purpose is choosing the attributes that have the percentage of missing under 5%:

```
cols = [var for var in df.columns if df[var].isnull().mean() < 0.05 and df[var].isnull().mean() > 0]
cols
#lọc cột miss 0-5%
```

```
['city_development_index',
 'enrolled_university',
 'education_level',
 'experience',
 'training_hours']
```

So now, we have the necessary attributes below:

```
df[cols].sample(5)
#trực quan data miss <5%
```

	city_development_index	enrolled_university	education_level	experience	training_hours
2245	0.926	no_enrollment	Primary School	20.0	18.0
6807	0.924	no_enrollment	Graduate	8.0	87.0
10463	0.926	no_enrollment	Graduate	12.0	15.0
7020	0.926	no_enrollment	Masters	20.0	58.0
7998	0.913	no_enrollment	Masters	12.0	158.0

+ Code

Then, calculating the percentage of records that have complete values for all columns in the list 'cols' with respect to the total number of rows in the DataFrame 'df':

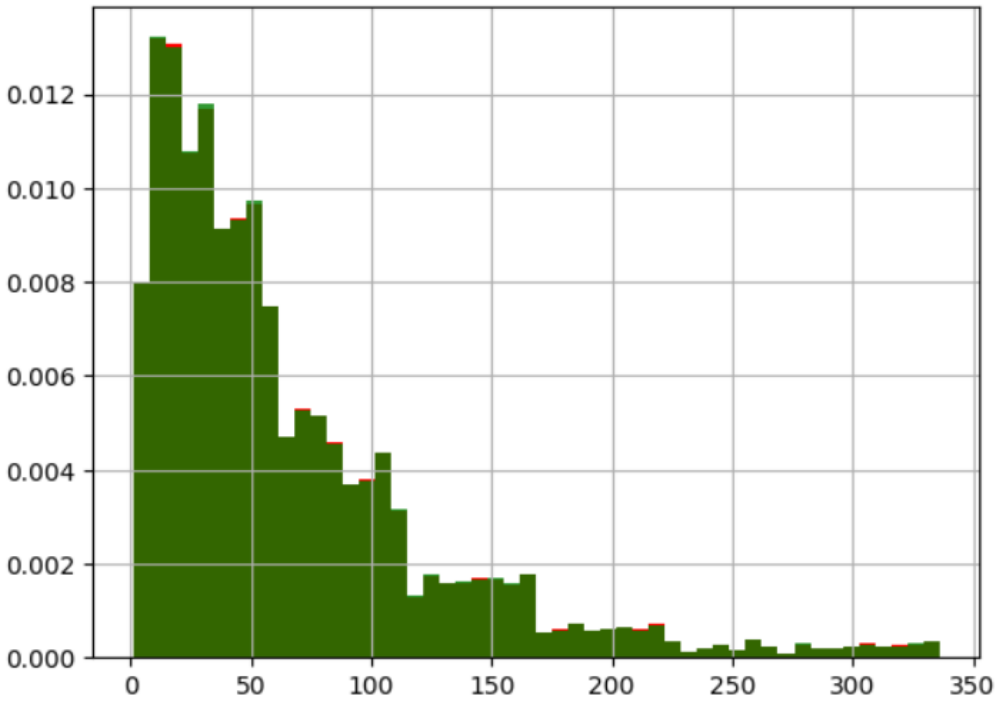
```
len(df[cols].dropna()) / len(df)
#tính tỷ lệ phần trăm các hàng (records)
# có đủ giá trị cho tất cả các cột nằm trong list 'cols' so với tổng số hàng trong DataFrame 'df'.
```

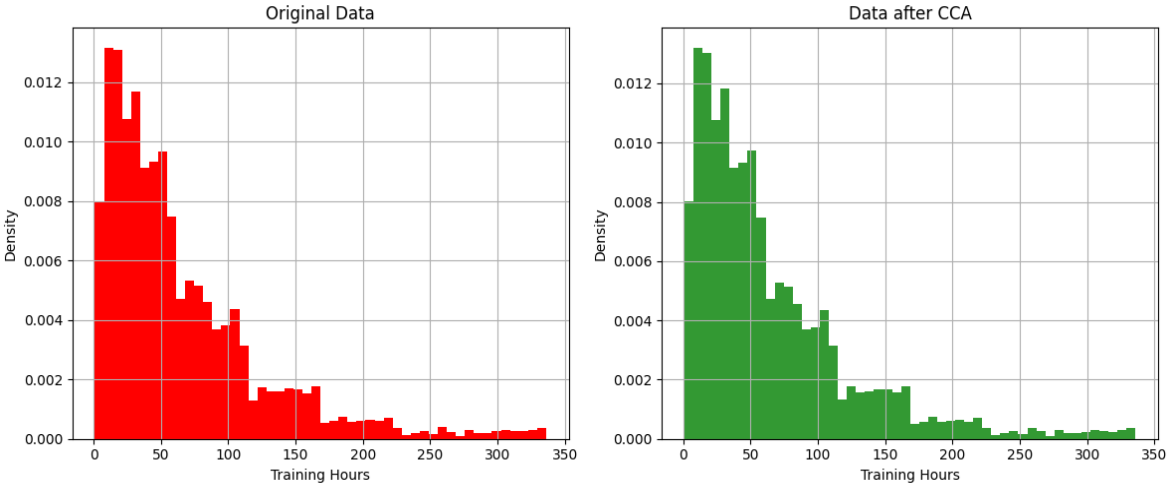
```
0.8968577095730244
```

The percentage after removing row approximately 0.89%
Next, we created a histogram visualization to compare the distribution of the 'training_hours' column between the original DataFrame 'df' and the DataFrame 'new_df' after applying the CCA using the source code below:

```
fig = plt.figure()
ax = fig.add_subplot(111)

# original data
df['training_hours'].hist(bins=50, ax=ax, density=True, color='red')

# data after cca, the argument alpha makes the color transparent, so we can
# see the overlay of the 2 distributions
new_df['training_hours'].hist(bins=50, ax=ax, color='green', density=True, alpha=0.8)
#green newdf
#red df
```

Then we have the result:





The histogram plots appear to be nearly identical, with the green bars representing the 'new_df' (data after CCA imputation) and the red bars representing the 'df' (original data).

And we do the same with other attributes:

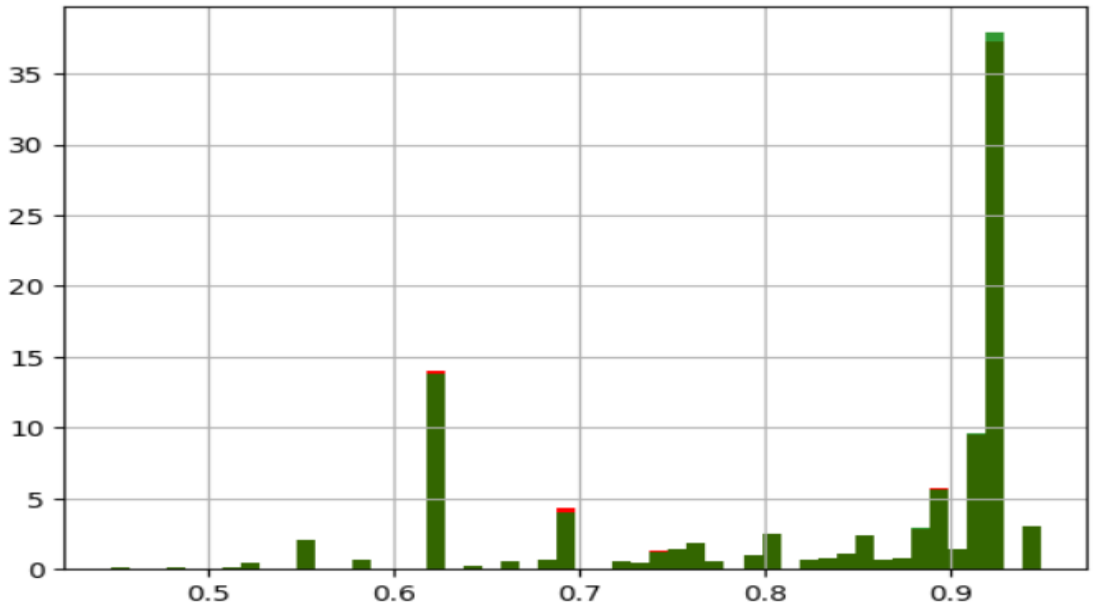


Figure 4: city_development_index(1)

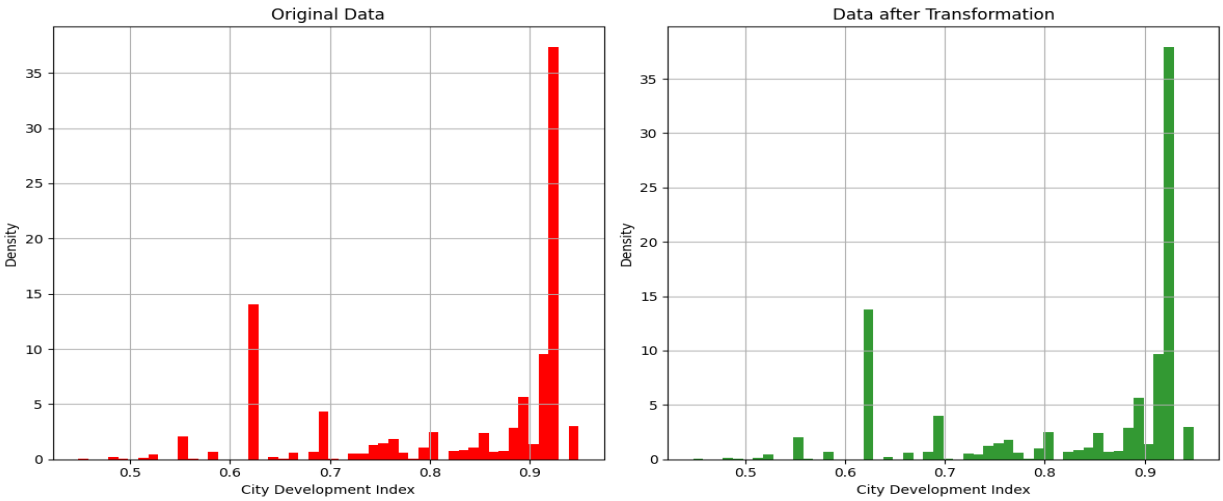


Figure 5: city_development_index(2)

```
temp = pd.concat([
            # percentage of observations per category, original data
            df['education_level'].value_counts() / len(df),

            # percentage of observations per category, cca data
            new_df['education_level'].value_counts() / len(new_df)
        ],
        axis=1)

# add column names
temp.columns = ['original', 'cca']

temp
```

	original	cca
Graduate	0.605387	0.619835
Masters	0.227633	0.234082
High School	0.105282	0.107380
Phd	0.021610	0.022116
Primary School	0.016077	0.016587

Figure 5: comparison in ‘educational_level’ after using CCA

Finally, applying CCA method for all the data set.

The performance of the CCA Imputation method was evaluated in conjunction with other missing data handling techniques to gauge its

efficiency and appropriateness for our specific dataset and analytical objectives.

3.2. Multivariate imputation:

Missing data can arise due to various reasons, such as survey non-response, data entry errors, or technical issues. To address this issue, researchers and data scientists have developed different techniques for imputing missing values. One such powerful approach is multivariate imputation, which leverages the relationships between variables to fill in the missing data. In this report, we will explore two popular methods of multivariate imputation: K-Nearest Neighbors (KNN) imputation and Iterative imputation.

3.2.1. KNN imputation

K-Nearest Neighbors imputation is a simple yet effective technique for filling in missing values in a dataset. The basic idea behind KNN imputation is to use the values of "k" nearest neighbors of a data point with missing values to estimate the missing entry. In this method, the similarity between data points is determined using distance metrics such as Euclidean distance or Manhattan distance. The "k" nearest data points to the one with missing values are then used to calculate a weighted average or majority vote to impute the missing value.

In order to determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated. These distance metrics help to form decision boundaries, which partitions query points into different regions. You commonly will see decision boundaries visualized with Voronoi diagrams.

While there are several distance measures that you can choose from, this article will only cover the following:

In the presence of missing coordinates, the Euclidean distance is calculated by ignoring the missing values and scaling up the weight of the non-missing coordinates.

$$d_{xy} = \sqrt{weight \ * squared \ distance \ from \ present \ coordinates}$$

where,

$$weight = \frac{Total\ number\ of\ coordinates}{Number\ of\ present\ coordinates}$$

KNN Imputer is used to handle missing values in the dataset. Then n_neighbors parameter is set to 3, which means the imputation is done based on the three nearest neighbors. The weights parameter is set to 'distance', indicating that the closer neighbors have a higher influence on the imputation. KNN imputation is a technique that replaces missing values with the weighted average of their nearest neighbors' values.

3.2.2. Iterative imputation (Round-robin linear regression)

Iterative Imputation, also known as Multiple Imputation by Chained Equations (MICE), is a popular method for dealing with missing data in statistical analysis and machine learning. It is an iterative approach that imputes missing values by creating multiple imputed datasets based on the observed data. One of the techniques used within MICE is Round-Robin Linear Regression.

Iterative Imputation using Round-Robin Linear Regression is used to handle missing values in the dataset. The term "Round-Robin" implies that the imputation is done in a cyclical manner, where each column with missing values is imputed in a sequence, and the process is repeated until convergence or a predefined number of iterations. The missing values in each column are predicted using linear regression models that are trained on the available data in other columns.

The imputation process then involves multiple iterations, where each column with missing values is sequentially imputed using linear regression models.

Data Preprocessing:

The code drops several columns from the DataFrame using df.drop(columns=[...], inplace=True), removing unnecessary features for the analysis.

	city_development_index	experience	training_hours
4762	0.920	9.0	13.0
5393	0.743	7.0	6.0
8122	0.926	6.0	118.0
145	0.926	20.0	120.0
8194	0.920	20.0	67.0

Missing Value Imputation using Linear Regression:
For each of the columns with missing values (city_development_index, experience, and training_hours), the missing values are replaced with the mean of their respective columns using fillna().

	city_development_index	experience	training_hours
4762	0.920	9.0	13.00
5393	0.923	7.0	6.00
8122	0.926	6.0	118.00
145	0.926	10.5	120.00
8194	0.920	20.0	64.25

A random seed is set using 'np.random.seed()' to ensure reproducibility.

A random sample of 5 rows is selected from the DataFrame using 'df.sample(5)'.

The last column of the DataFrame is removed using 'df = df.iloc[:, 0:-1]'.

Specific elements in the DataFrame are set to 'NaN' (missing value) artificially using 'np.NaN'.

The missing values are then imputed using a three-step iterative process. For each iteration:

- A new DataFrame (e.g., df0, df1, df2) is created for storing the imputed values.

- The missing values in the 'city_development_index', 'experience', and 'training_hours' columns are filled with the mean of the respective columns.

- Linear Regression models are trained using the known values for each row and used to predict the missing values iteratively. The trained models are then used to predict the missing values and update the DataFrame accordingly.

Final Missing Value Imputed DataFrame:

The process of iterative imputation continues until the missing values are imputed in all columns.

The last DataFrame 'df3' represents the final result after all missing values have been imputed using linear regression.

Comparing Iterations:

After each iteration, the code calculates the difference between the current DataFrame and the one from the previous iteration to observe the changes introduced by the imputation.

Final Output:

The code displays the resulting DataFrame df3 after all three iterations, which should now have imputed values for all the missing entries.

	city_development_index	experience	training_hours
4762	0.920	9.00	13.0
5393	0.920	7.00	6.0
8122	0.926	6.00	118.0
145	0.926	6.57	120.0
8194	0.920	20.00	51.5

The Iterative Imputation (using Round-Robin Linear Regression) seems to follow an iterative approach to impute missing values using linear regression models, but it could be challenging to determine the accuracy

and reliability of this imputation method without further context about the dataset and the nature of the missing values. Additionally, this iterative imputation technique may not be the most efficient way to handle

missing values, and other imputation methods such as K-nearest neighbors or mean/median imputation might be more appropriate depending on the dataset's characteristics.

3.2.3. Arbitrary value imputation

Arbitrary value imputation is a type of data imputation technique used in machine learning to fill in missing values in datasets. It involves replacing missing values with a specified arbitrary value, such as 0, 99, 999, or negative values. Instead of imputing the numbers using statistical averages or other methods, the main goal is to flag the values.

Some advantages of arbitrary value imputation:

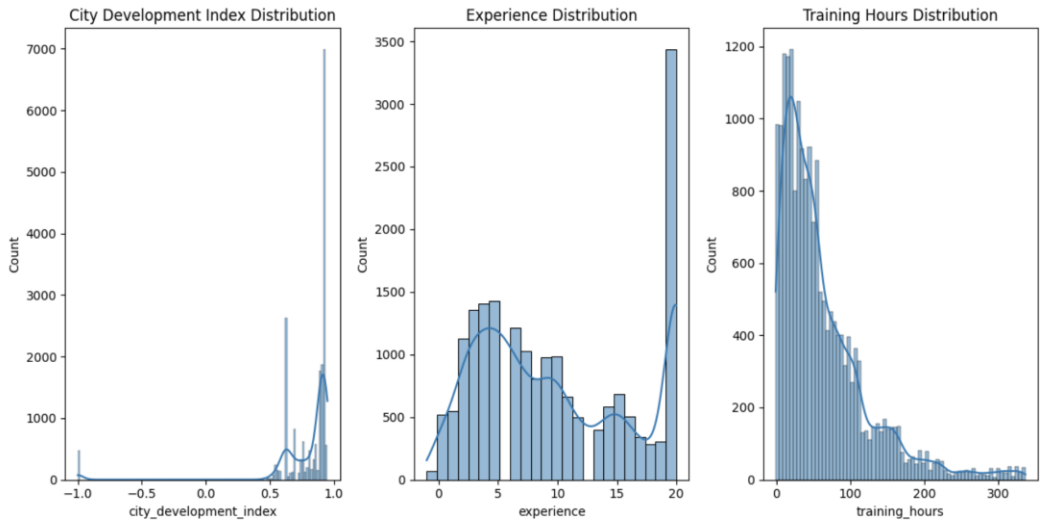
- Simplicity: Arbitrary value imputation is easy to implement and does not require complex algorithms and compuations. It's a straightforward way to address missing data.
- Usefulness for non-statistical data: Arbitrary value imputation can be useful when dealing with categorical or non-numeric data where statistical methods might not be as applicable.

In the scope of our project, we use -1 as arbitrary values for 'City_development_index', 'Experience' and 'Training_hours' respectively.

```
In [65]: print('Original City development index variable variance: ', df['city_development_index'].var())
         print('City development index Variance after -1 imputation: ', df['city_development_index'].fillna(-1).var())
         print()
         print('Original Experience variable variance: ', df['experience'].var())
         print('Experience Variance after -1 imputation: ', df['experience'].fillna(-1).var())
         print()
         print('Original Training hours variable variance: ', df['training_hours'].var())
         print('Training hours Variance after -1 imputation: ', df['training_hours'].fillna(-1).var())

         Original City development index variable variance:  0.015211374664424446
         City development index Variance after -1 imputation:  0.09637945318893501

         Original Experience variable variance:  42.31851676548554
         Experience Variance after -1 imputation:  42.57875524175738

         Original Training hours variable variance:  3586.2881933989347
         Training hours Variance after -1 imputation:  3611.0439874323233
```

When using arbitrary value imputation technique, we use kernel density estimation(kde) using Gaussian as kernel function to check the variance and distribution of data points before and after imputed.

And here is the results we chieved after computing the data.



3.2.4. Frequent value imputation

Frequent value imputation, also known as mode value imputation, is a technique used to fill in missing values in a dataset by replacing them with the most common value (mode) of the respective feature. When dealing with real-world datasets, it is common to encounter missing data, which can be problematic when performing data analysis or training machine learning models since many algorithms cannot handle missing values.

The frequent value imputation method is simple yet effective, especially for categorical or discrete features. When a missing value is encountered for a particular feature, the missing data point is replaced with the value that occurs most frequently in that specific feature's column.

3.2.5. Mean and median value imputation

Mean value imputation involves replacing missing values with the mean (average) value of the non-missing values in the same feature (column). This technique assumes that the missing values are missing at random and that the mean is a representative value for the entire distribution of the feature.

Median value imputation is similar to mean value imputation, but instead of using the mean, you use the median value of the non-missing values.

The median is the middle value in a dataset when it's sorted in ascending order. This technique is often preferred when the feature's distribution is skewed or contains outliers because the median is less sensitive to extreme values than the mean.

```python
X_train['City_development_index_median'] = X_train['city_development_index'].fillna(median_cdi)
X_train['City_development_index_mean'] = X_train['city_development_index'].fillna(mean_cdi)

X_train['Experience_median'] = X_train['experience'].fillna(median_exp)
X_train['Experience_mean'] = X_train['experience'].fillna(mean_exp)

X_train['Training_hours_median'] = X_train['training_hours'].fillna(median_hour)
X_train['Training_hours_mean'] = X_train['training_hours'].fillna(mean_hour)
```

Python

4. Conclusion

Missing data is a real-world challenge that can be encountered in various practical scenarios across different fields. Whether it's analyzing customer data in marketing, medical records in healthcare, or survey responses in social sciences, missing data can undermine the quality and reliability of conclusions drawn from the analysis.

In practical terms, addressing missing data requires a thoughtful approach that strikes a balance between preserving valuable information and avoiding biased results. Complete case analysis, while simple, may not be the best choice, as it often leads to data loss and potential skewing of results. Instead, employing appropriate imputation techniques, such as mean imputation for numerical data or mode imputation for categorical data, can help fill in missing values based on existing information. For more complex datasets and scenarios, advanced modeling techniques like multiple imputation or the use of machine learning algorithms can yield better imputation results. By creating multiple plausible imputed datasets and incorporating the uncertainty into the analysis, researchers and practitioners can enhance the robustness of their conclusions.

In terms of future scope, the field of missing data handling continues to evolve, and new methods are being developed to tackle this issue more effectively. Researchers are exploring novel techniques that can better capture the complexity of missing data patterns, especially in high-dimensional datasets and complex models. Machine learning approaches, such as Generative Adversarial Networks (GANs) and deep learning

architectures, show promise in imputing missing data and may offer more accurate predictions. Additionally, the integration of domain-specific knowledge and expert insights into the imputation process could lead to more context-aware and meaningful imputations.

In conclusion, addressing missing data practically involves adopting appropriate imputation techniques while considering the context of the dataset and the implications of the missing data mechanism. With ongoing research and technological advancements, the future scope for handling missing data looks promising, offering more sophisticated and accurate methods to derive reliable conclusions from incomplete datasets. However, it remains essential for researchers and analysts to be vigilant about the potential biases and limitations that missing data can introduce and to employ transparent and well-documented methods in their analyses.

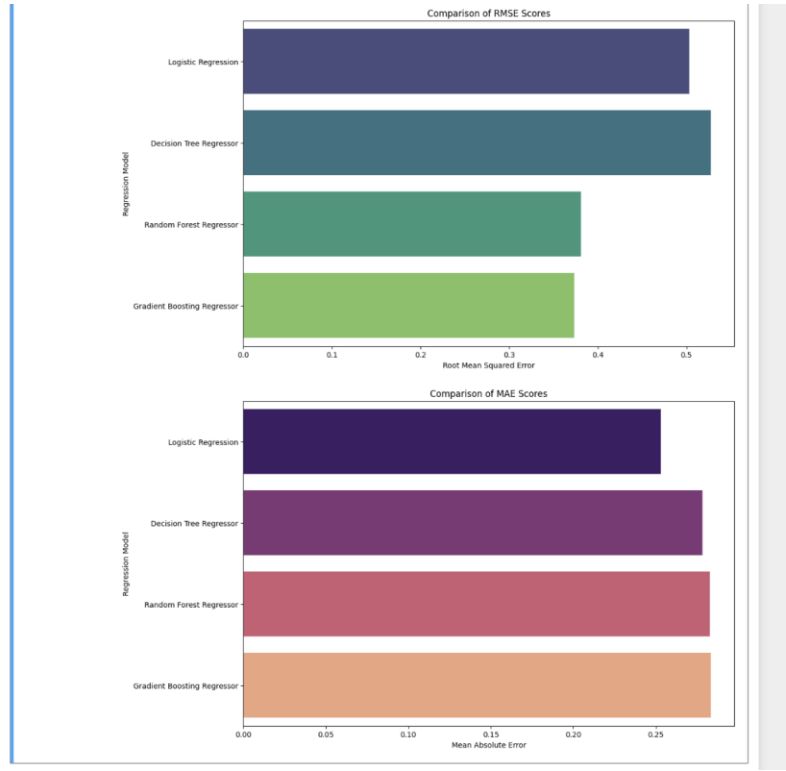
4. Evaluation

We use 4 models: Logistic regression, Decision Tree Regressor, Random Forest Regressor and Gradient Boosting to test the efficiency of the imputing methods.

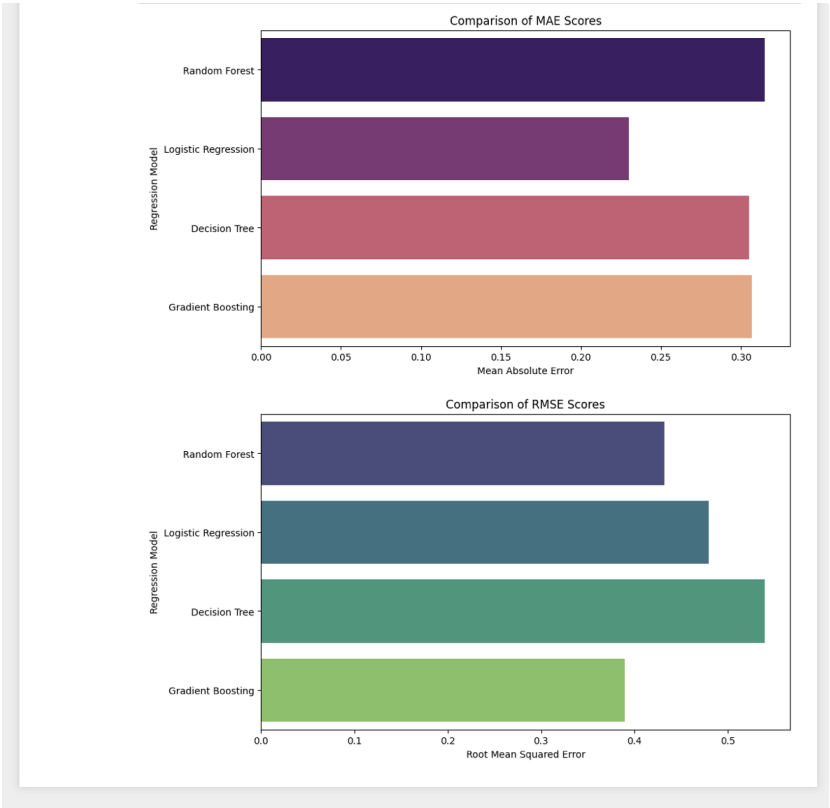
We also use Root mean square errors (RMSE) and Mean absolute errors (MAE) as measures for the performance of the models.

And here are the results:

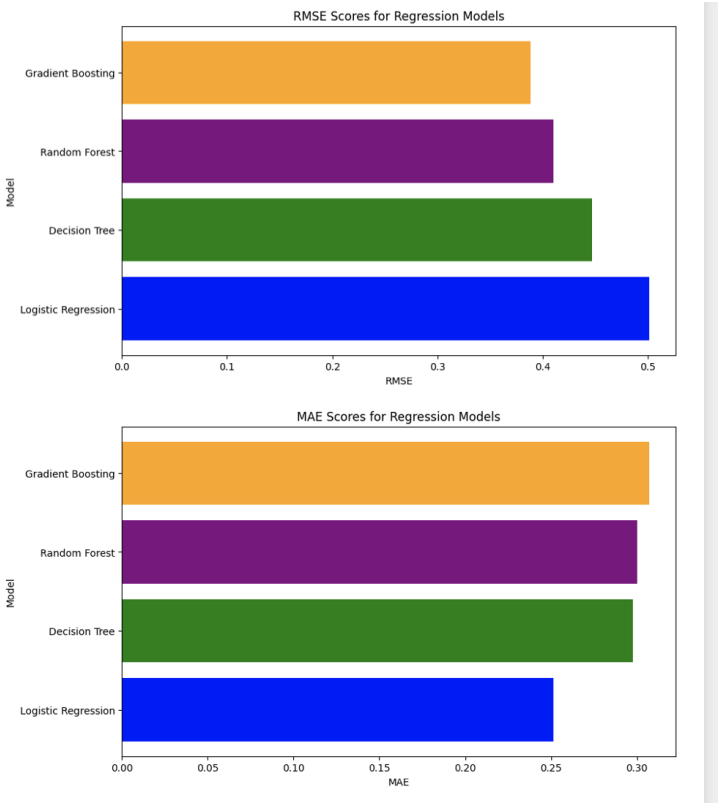
- Arbitrary imputation:



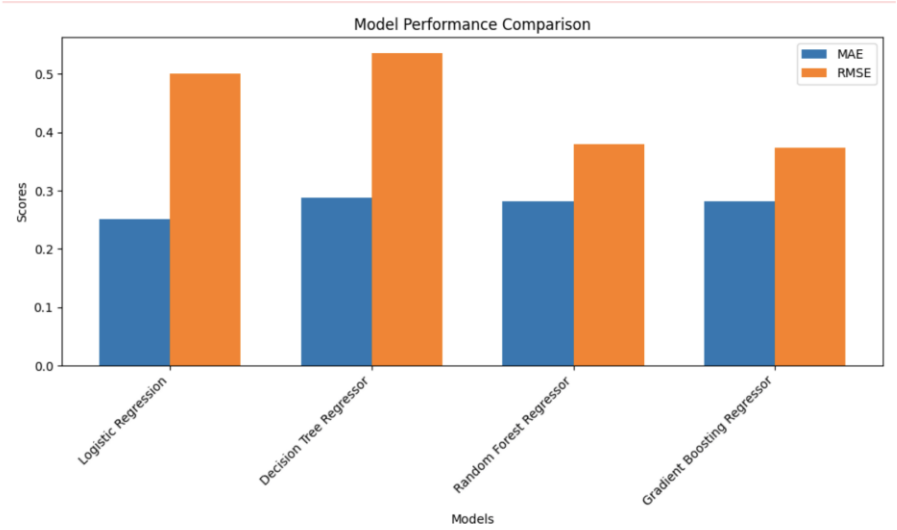
- Complete case analysis:



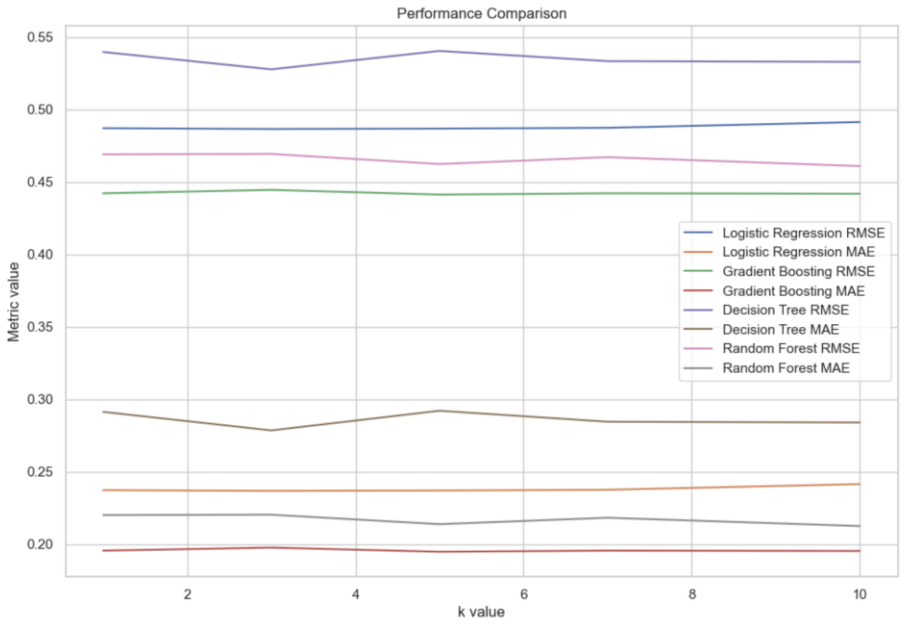
- Frequent vaclue imputation:



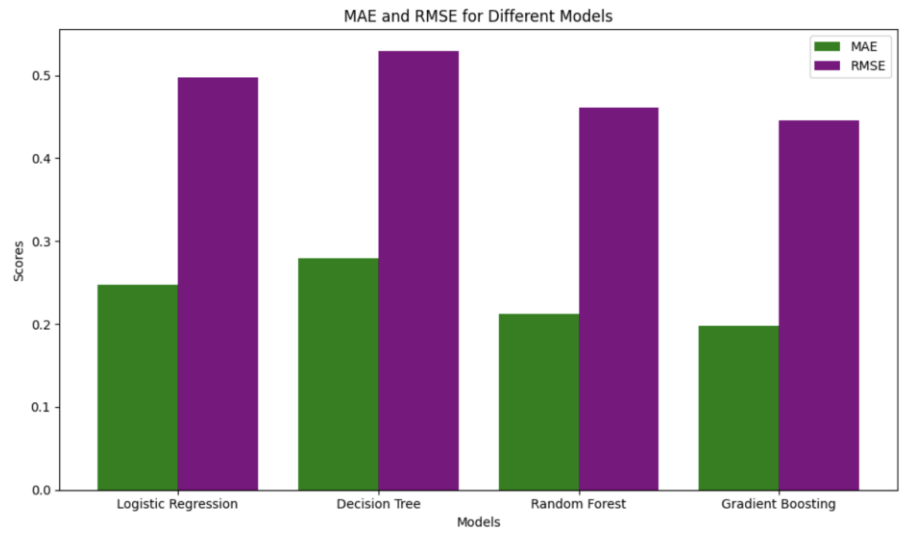
- Iterative imputing:



- KNN imputation:



- Mean value imputation:



5. References

[1] https://www.kaggle.com/code/parulpandey/a-guide-to-handling-missing-values-in-python

[2] https://www.analyticsvidhya.com/blog/2020/07/knnimputer-a-robust-way-to-impute-missing-values-using-scikit-learn/

[3] https://towardsdatascience.com/a-better-way-to-handle-missing-values-in-your-dataset-using-iterativeimputer-9e6e84857d98

[4]

https://pubmed.ncbi.nlm.nih.gov/22218574/

[5] https://github.com/campusx-official/100-days-of-machine-learning/tree/main

[6] Dataset: https://github.com/campusx-official/100-days-of-machine-learning/blob/main/day35-complete-case-analysis/data_science_job.csv