Python Technical Report

Jiji Ajitha Kumari Venugopalan Assari

Student ID - 20033188

Contents

1 Introduction

The report covers the analysis of a dataset based on real-world Islington iWork unemployment support of the year 2018-2021. We will be discussing about data understanding, preparation, exploration, analysis and data mining in the further parts. The report will illustrate the output along with the screenshots of the code from the analysis. The main objective of the study is to prepare the data for further mining and analysis.

2 Data Understanding

The dataset consists of 4788 records of 14 variables including socio-demographic and personal information. Employer is our target variable and we need to decide on the explanatory variable for further analysis.

```python
#Read and store a csv file into the dataframe
Islington_data = pd.read_csv('Islington_iwork_anonymous_data.csv',sep=',')

#Display the first five rows of the dataframe
Islington_data.head()
```

Figure 1: :Importing the data

Out[189]:

	Employer	Registration_Date	Client_Current_Age	Parent_on_Enrolment	Gender	Ethnic_Origin	Has_Disability	Disability_det
0	No Outcome	22/07/2021	29	Blanks	Female	(C) Asian or Asian British - Any other Asian b...	No	N
1	No Outcome	24/08/2021	32	Blanks	Male	(C) Asian or Asian British - Any other Asian b...	No	N
2	No Outcome	13/05/2021	48	Blanks	Female	(D) Black or Black British - Other African	Blanks	N
3	No Outcome	31/08/2021	55	Blanks	Male	(D) Black or Black British - Any other Black b...	No	N
4	No Outcome	31/08/2021	30	Blanks	Female	(A) White - Any other White background	No	N

Figure 2: :Displaying the sample data

```
In [190]:
#To print the shape of the table
print("number of rows in the data is", len(Islington_data))
print("number of columns in the data is", len(Islington_data.columns))
```

```
number of rows in the data is 4788
number of columns in the data is 14
```

Figure 3: : Shape of the Data

2.1 Creating a metadata Table

The metadata was created using SPSS. Since NULL is considered as a string in SPSS, it is not considered as a missing value. The variable Client_Current_Age is the only one which has numeric measure and all the other variables are Nominal with type String. Employer is set as target manually and all the other variables are considered as input by the software. If we need to change any data, we can manually adjust the data as per the requirement.

	Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure	Role
1	Employer	String	48	0		None	None	48	Left	Nominal	Target
2	Registration_Date	String	10	0		None	None	10	Left	Nominal	Input
3	Client_Current_Age	Numeric	3	0		None	None	8	Right	Scale	Input
4	Parent_on_Enrolment	String	6	0		None	None	6	Left	Nominal	Input
5	Gender	String	17	0		None	None	17	Left	Nominal	Input
6	Ethnic_Origin	String	55	0		None	None	26	Left	Nominal	Input
7	Has_Disability	String	17	0		None	None	10	Left	Nominal	Input
8	Disability_details	String	4	0		None	None	10	Left	Nominal	Input
9	Religion	String	26	0		None	None	26	Left	Nominal	Input
10	Sexuality	String	27	0		None	None	27	Left	Nominal	Input
11	Highest_Level_of_Education	String	58	0		None	None	28	Left	Nominal	Input
12	Claiming_Benefits	String	6	0		None	None	6	Left	Nominal	Input
13	Benefits	String	105	0		None	None	26	Left	Nominal	Input
14	WARD_NAME	String	31	0		None	None	31	Left	Nominal	Input
15											
16											
17											
18											
19											
20											

Figure 4: : MetaData

2.2 Checking missing or error data of each variable

The number of null values for each variable is displayed in Figure 5. The output shows that all the values in Disability_details are null and 2507 of Benefits are also null. There is 1 missing value in Employer and 7 in WARD_NAME.

```
#For missing values
for i in Islington_data.columns:
    print(f"The number of missing values in {i} is {Islington_data[i].isna().sum()}")
```

```
The number of missing values in Employer is 1
The number of missing values in Registration_Date is 0
The number of missing values in Client_Current_Age is 0
The number of missing values in Parent_on_Enrolment is 0
The number of missing values in Gender is 0
The number of missing values in Ethnic_Origin is 0
The number of missing values in Has_Disability is 0
The number of missing values in Disability_details is 4788
The number of missing values in Religion is 0
The number of missing values in Sexuality is 0
The number of missing values in Highest_Level_of_Education is 0
The number of missing values in Claiming_Benefits is 0
The number of missing values in Benefits is 2507
The number of missing values in WARD_NAME is 7
```

Figure 5: : Missing Values in each variable

3 Data Preparation

3.1 To reduce the variables

The missing values shown in 5 reveals that all the entries in Disability_details is null values and majority of values in Benefits are null values. We need to remove two of the variables as we need to drop the rows with null values to prepare the data for analysis. If Benefits is not removed, we will lose 2507 records when dropping the rows.

The variable registration date can be removed as it does not provide any relevant information. If there was exact hiring date, we could introduce a new variable showing how long it took to get the job. However, this is not the case here. If we check the frequency of unique values in sexuality as in 6, most people are heterosexual. If we consider this as an explanatory variable, there is a chance that it may serve as a bias.

```
In [193]: Islington_data['Sexuality'].value_counts()
          #Since most of the person appears to be Hetrosexual, this variable
          #cannot be considered as a fit to predict the target variable
          #as it may serve as a bias.

Out[193]: Heterosexual                 3592
          Prefer not to say sexuality   552
          No response to sexuality      198
          Blanks                        195
          Gay / lesbian                 125
          Bisexual                      107
          Other sexuality                19
          Name: Sexuality, dtype: int64
```

Figure 6: : Frequency of unique values in Sexuality

The mentioned columns are then removed from the data for further analysis.

In [195]:
```
# Remove the irrelevant columns
Islington_data=Islington_data.drop(['Disability_details', 'Registration_Date',\
                'Sexuality','Benefits'], axis = 1)
Islington_data.head()
```

Out[195]:

	Employer	Client_Current_Age	Parent_on_Enrolment	Gender	Ethnic_Origin	Has_Disability	Religion	Highest_Level_of_Educ
0	No Outcome	29	Blanks	Female	(C) Asian or Asian British - Any other Asian b...	No	Blanks	ISCED Level 6 (Bachelo equivalent
1	No Outcome	32	Blanks	Male	(C) Asian or Asian British - Any other Asian b...	No	Blanks	ISCED Level 6 (Bachelo equivalent
2	No Outcome	48	Blanks	Female	(D) Black or Black British - Other African	Blanks	Christian	B
3	No Outcome	55	Blanks	Male	(D) Black or Black British - Any other Black b...	No	Blanks	ISCED Level 7 (Maste equivalent
4	No Outcome	30	Blanks	Female	(A) White - Any other White background	No	Blanks	ISCED Level 2 (L secondary educ

Figure 7: : Data after removing irrelevant variables

3.2 To clean data

Since there are missing values in Employer and in WARD_NAME, the rows corresponding to these values should be removed. This is done using the function dropna().

In [199]:
```
#Removing the rows with null values
Islington_data=Islington_data.dropna()
```

Figure 8: : Removing rows with null values

3.3 To transform variables

The objective is to transform 6 variables into ordinal values. The target variable Employer is transformed such that No Outcome must be assigned as 0 and 1 must be assigned to those rows with an outcome. 4179 of the employers has no outcome and 601 has an outcome as shown in the Figure 9.

```
#Transform the variable Employer

Islington_data['Employer']=Islington_data['Employer'].apply(lambda x: 0 if x == 'No Outcome' else 1)
Islington_data['Employer'].value_counts()
```

```
0    4179
1     601
Name: Employer, dtype: int64
```

Figure 9: : Transforming Employer into ordinal values

The Gender variable is transformed into ordinal where 0 represents Female, 1 represents Male, 2 for Transgender, 3 for Prefer not to say and 4 for any others.

```
#Transform the variable Gender

Islington_data['Gender']=Islington_data['Gender'].apply(lambda x: 0 if x=='Female' else
                                                 else 3 if x=='Prefer not to say' else 4)
Islington_data['Gender'].value_counts()
```

```
0    2937
1    1784
3      33
4      20
2       6
Name: Gender, dtype: int64
```

Figure 10: : Transforming Gender into ordinal values

Ethnic origin is transformed based on the occurence in the data set in descending order. Any entry starting with (E) is assigned 1 , (D) as 2, (C) as 3, (B) as 4,(A) as 5 and else 0.

```
#getting unique values and the ordinal values
values = Islington_data['Ethnic_Origin'].value_counts().keys().tolist()
indx=range(len(values))
valdict=dict(zip(values,indx))
```

```
#Transform the variable Ethnic Origin

Islington_data['Ethnic_Origin']=Islington_data['Ethnic_Origin'].apply(lambda x: valdict[x])

```

```
0     1130
```

Figure 11: : Transforming Ethnic_Origin into ordinal values

The WARD_NAME is considered and transformed into ordinal based on their occurence in ascending order.

```
#getting unique values sorted in ascending order and the ordinal values
values2 = Islington_data['WARD_NAME'].value_counts(ascending=True).keys().tolist()
indx2=range(len(values2))
valdict2=dict(zip(values2,indx2))
```

```
#Transform the variable WARD_NAME

Islington_data['WARD_NAME']=Islington_data['WARD_NAME'].apply(lambda x: valdict2[x])
```

Figure 12: : Transforming WARD_NAME into ordinal values

The Figure 13 shows the transformation of Highest_Level_if_Education based on UK ISCED Level

```
#Transform the variable Highest_Level_of_Education

Islington_data['Highest_Level_of_Education']=Islington_data['Highest_Level_of_Education'].apply(lambda x: \
                                                     1 if x.startswith('ISCED Level 8')\
                                                     else 2 if x.startswith('ISCED Level 7')\
                                                     else 3 if x.startswith('ISCED Level 6')\
                                                     else 4 if x.startswith('ISCED Level 5')\
                                                     else 5 if x.startswith('ISCED Level 4')\
                                                     else 6 if x.startswith('ISCED Level 3')\
                                                     else 7 if x.startswith('ISCED Level 2')\
                                                     else 8 if x.startswith('ISCED Level 1')\
                                                     else 9 if x.startswith('ISCED Level 0') else 0)

Islington_data['Highest_Level_of_Education'].value_counts()
```

```
0    2680
7     632
6     463
3     390
5     183
8     173
2     105
4      96
9      49
1       9
Name: Highest_Level_of_Education, dtype: int64
```

Figure 13: : Transforming Highest_Level_if_Education into ordinal values

Claiming Benefits are converted into ordinal numbers with 0 for No, 1 for Yes and 2 for Blank.

```
#Transform the variable Claiming_Benefits

Islington_data['Claiming_Benefits']=Islington_data['Claiming_Benefits'].apply(lambda x: 0 if x=='No' \
                                                                              else 1 if x=='Yes' else 2)
Islington_data['Claiming_Benefits'].value_counts()

```

```
2    2647
1    1324
0     809
Name: Claiming_Benefits, dtype: int64
```

Figure 14: : Transforming Claiming_Benefits into ordinal values

We remove all the columns without a numeric values from our data.

```
#Removing all the other columns without ordinal values
Islington_data_new=Islington_data.drop(['Parent_on_Enrolment', 'Has_Disability',\
                 'Religion'], axis = 1)
```

```
Islington_data_new.head()
```

	Employer	Client_Current_Age	Gender	Ethnic_Origin	Highest_Level_of_Education	Claiming_Benefits	WARD_NAME
0	0	29	0	10	3	0	100
1	0	32	1	10	3	0	120
2	0	48	0	3	0	2	117
4	0	30	0	1	7	1	119
5	0	57	1	0	3	1	109

Figure 15: : Dropping columns without numeric values

4 Data Analysis

4.1 Summary statistics of age variable

Figure 16 shows the descriptive statistics for age variable. The mean is 37.32 and variance is given as 14.08. Skewness is 0.906 and kurtosis is 3.205.

```
summary=Islington_data_new['Client_Current_Age'].describe()
summary.loc['skewness'] = Islington_data_new['Client_Current_Age'].skew().tolist()
summary.loc['kurtosis'] = Islington_data_new['Client_Current_Age'].kurtosis().tolist()
summary
```

```
count       4780.000000
mean          37.321130
std           14.083748
min            0.000000
25%           26.000000
50%           36.000000
75%           47.000000
max          137.000000
skewness       0.905882
kurtosis       3.205191
Name: Client_Current_Age, dtype: float64
```

Figure 16: : Summary of age variable

4.2 Correlation of each variable with target variable

The correlation matrix is found and the correlation plot is given in the code. However, the correlation between the explanatory variables and Employer is the required table as displayed in 18.

```
#Find correlation matrix
corr_matrix = Islington_data_new.corr()
```

Figure 17: : Finding correlation matrix

We can find the variables that are most correlated to the target variable by considering absolute value of the correlation. corr_matrix[:1].abs().unstack().sort_values(ascending=FALSE) can help to find the top correlated variables.

```
#correlation of target variable with other variables
corr_matrix[:1].unstack().sort_values(ascending = False)[1:]
```

```
Highest_Level_of_Education  Employer    0.293205
Gender                      Employer    0.021403
WARD_NAME                   Employer    0.007149
Ethnic_Origin               Employer   -0.006238
Client_Current_Age          Employer   -0.069318
Claiming_Benefits           Employer   -0.261334
dtype: float64
```

Figure 18: : Correlation of each variable with target variable

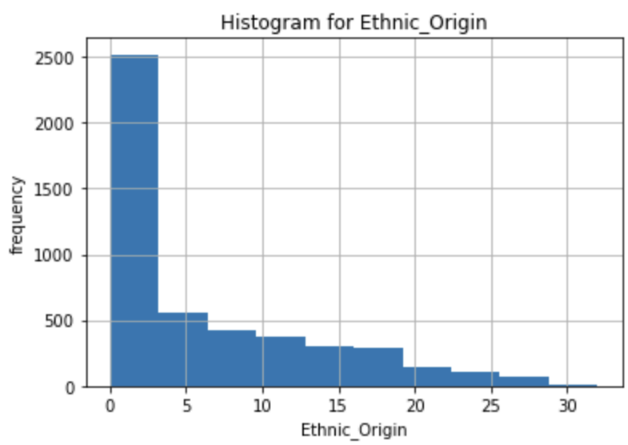
5 Data Exploration

5.1 Histogram plot for any user chosen variable until exit

The code in 19 allows to continue running until the user exits by pressing enter without giving any input. The entered variable is stripped of any spaces at the extreme ends of the input string and if the input is not empty, the code creates a histrogram with appropriate title along with x and y labels.

```python
while True:
    #convert user-input it into a string and store in variable
    variable = str(input("Enter the variable to show the histogram:"))
    #remove any spaces at the extreme ends of the input
    variable=variable.strip()
    #Stop if no input is given
    if variable == '':
        print("You entered blank")
        break
    else:
        #plot the histogram for the input
        df=Islington_data_new[variable]
        ax = df.hist(figsize=(6,4))
        plt.title(f"Histogram for {variable}")
        plt.xlabel(f"{variable}")
        plt.ylabel("frequency")
        plt.show()
        continue
```

```
Enter the variable to show the histogram:Ethnic_Origin
```




```
Enter the variable to show the histogram: |
```

Figure 19: : Histogram plot for user-input variable

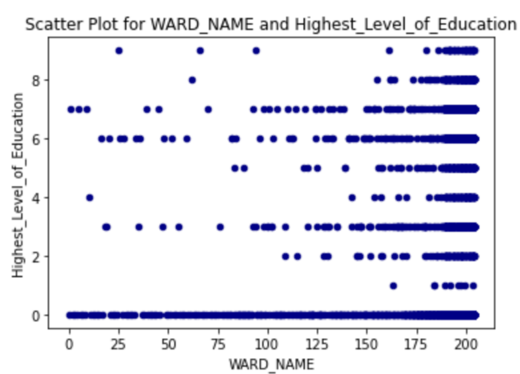
5.2 Scatter plot for any user chosen variables until exit

The process is similar to that in the previous section. However, two inputs should be given to display a scatter plot with the input variables. If the user does not enter any variable into the textbox, the process terminates else it asks for the second variable. Again, if no input is given, the process ends.

```python
while True:
    #convert user-input it into a string and store in variable
    x= str(input("Enter the first variable to show the scatter plot \n"))
    #remove any spaces at the extreme ends of the input
    x=x.strip()
    #Stop if no input is given
    if x=='':
        print("The first variable was empty")
        break
    #convert user-input it into a string and store in variable
    y=str(input("Enter the second variable to show the scatter plot \n"))
    #remove any spaces at the extreme ends of the input
    y=y.strip()
    #Stop if no input is given
    if y == '':
        print("Cannot plot a scatter diagram as the second variable was not given")
        break
    else:
        #plot the scatter plot for the input variables
        ax = Islington_data_new.plot.scatter(x=x,y=y, c='DarkBlue')
        plt.title(f"Scatter Plot for {x} and {y}")
        plt.xlabel(f"{x}")
        plt.ylabel(f"{y}")
        plt.show()
        continue
```

Figure 20: : Code for Scatter plot for user-input variable

```
Enter the first variable to show the scatter plot
WARD_NAME
Enter the second variable to show the scatter plot
Highest_Level_of_Education
```




```
Enter the first variable to show the scatter plot
```

Figure 21: : Scatter plot for user-input variable

The scatter plot for WARD_NAME against Highest_Level_if_Education is plotted in Figure 21

6 Data Mining

The explanatory variable is standardised such that mean is 0 and variance is 1 to get a better fitting model. This to make our data consistent. After standardising the data, the data is split into training and testing data. 70% of the data is considered as training data and 30% as testing data.

```
from sklearn.preprocessing import StandardScaler

#Defining X as the explanatory variables and Y as target variable
X=Islington_data_new.drop(columns=['Employer'])
Y=Islington_data_new['Employer']

scaler = StandardScaler()

#Standardise X such that mean=0 and variance=1
X = scaler.fit_transform(X)

#Splitting into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=50)
```

Figure 22: : Training and Testing data

6.1 Predictive models to predict client employment using the prepared variables

The two predictive models that we consider are KNeighborsClassifier and RandomForestClassifier. The data is fitted using these two models and predicted values are derived for testing data in both cases.

```
#import KNeighborsClassifier and accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

#To initiate the classifier
model_KN = KNeighborsClassifier()

# fit classifier to training set
model_KN.fit(X_train,y_train)

#predict the values of testing data
predictions = model_KN.predict(X_test)
```

Figure 23: : KNeighborsClassifier

True Positive(TP) indicates the outcomes which were correctly predicted by the model in the positive clases whereas true negative(TN) is an outcome where the model correctly predicts the negative class. False Positive(FP) is the number of wrongly detected positive outcomes. False Negative(FN) is the number of wrongly detected negative outcomes.

```python
from sklearn.metrics import confusion_matrix

#To find and assign the True Positives, True Negatives, False Positives and False Negatives
TN, FP, FN, TP = confusion_matrix(y_test, predictions).ravel()

#To print the values
print('True Positives:', TP)
print('False Positives:', FP)
print('True Negatives:', TN)
print('False Negatives:', FN)
```

```
True Positives: 40
False Positives: 55
True Negatives: 1203
False Negatives: 136
```

```python
#To compute and print the accuracy rate for training data
print('Training set score: {:.4f}'.format(model_KN.score(X_train, y_train)*100))

#To compute and print the accuracy rate for testing data
print('Test set score: {:.4f}'.format(model_KN.score(X_test, y_test)*100))
```

```
Training set score: 89.0915
Test set score: 86.6806
```

Figure 24: : Finding TP, FP, TN, FN & accuracy for the first model

The accuracy for training and testing data is comparably similar for the first model. The training data set accuracy was found to be 89.09 and that of the testing data was 86.68. This means that the model is performing quite well for the testing data equivalently as the training data.

```python
#import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

#To initiate the classifier
model_RF = RandomForestClassifier()

# fit classifier to training set
model_RF.fit(X_train, y_train)

#predict the values of testing data
predictions = model_RF.predict(X_test)
```

Figure 25: : RandomForest Classifier

The second model considers RandomForestClassifier. The accuracy as shown in Figure 26 is almost 99.7609 for training data. However, the accuracy is around 89.2608 for testing data.

```
#To find and assign the True Positives, True Negatives, False Positives and False Negatives
TN, FP, FN, TP = confusion_matrix(y_test, predictions).ravel()

#printing the values of TP, TN, FP & FN
print('True Positives:', TP)
print('False Positives:', FP)
print('True Negatives:', TN)
print('False Negatives:', FN)
```

```
True Positives: 48
False Positives: 26
True Negatives: 1232
False Negatives: 128
```

```
#To compute and print the accuracy rate for training data
print('Training set score: {:.4f}'.format(model_RF.score(X_train, y_train)*100))

#To compute and print the accuracy rate for testing data
print('Test set score: {:.4f}'.format(model_RF.score(X_test, y_test)*100))
```

```
Training set score: 99.7609
Test set score: 89.2608
```

Figure 26: : Finding TP, FP, TN, FN & accuracy for the second model

7 Conclusion

The data was analysed and cleaned for further data processing as the initial task. Then, two predictive model were fitted for training data and validated using the testing data. When comparing the True Positives, the second model identified about 48 correctly as opposed to 40 by the first model. False positive values are also higher for the first model when compared to the second. However, the first model has the accuracy rate of training model similar to the testing model. Identical accuracy rate is a good outcome as it means the model is showing almost the same efficiency as the training data set. Thus, the first model is a better fit than the second model.

Increasing the number of data will surely increase the accuracy of the model and the major issue that we need to focus is to avoid overfitting. Higher accuracy in training data may not mean better performance as it may lower the accuracy for testing data.

References

[1] Michael Bowles, *Machine Learning in Python*, Wiley, 2015

[2] Jonathan Adams,*Python Machine Learning from Scratch* , AI Sciences LLC, 2016

[3] Andreas C Muller, Sarah Guido,*Introduction to Machine Learning with Python* , O'Reily, 2016

[4] Rudolf Russel,*Machine Learning: Step-by-Step Guide To Implement Machine Learning Algorithms with Python* , 2018.

[5] Peters Morgan,*Data Analysis From Scratch With Python: Beginner Guide using Python, Pandas, NumPy, Scikit-Learn, IPython* , AI Sciences, 2018.

[6] Wes McKinney,*Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* , O'Reily, 2017.

[7] Jake VanderPlus,*Python Data Science Handbook* , O'Reily, 2016.

[8] Joel Grus,*Data Science from Scratch: First Principles with Python* , O'Reily, 2016.

[9] Bernd Klein,*Python Course Machine Learning With Python* .

[10] Chris Albon,*Machine Learning with Python Cookbook* , O'Reily, 2018.