

► ML Data Cleaning and Feature Selection

In this assignment, you will use a dataset for predictive learning and check the quality of the data and determine which features are important.

Answer the following questions:

- What are the data types? (Only numeric and categorical)
- Are there missing values?
- What are the likely distributions of the numeric variables?
- Which independent variables are useful to predict a target (dependent variable)? (Use at least three methods)
- Which independent variables have missing data? How much?
- Do the training and test sets have the same data?
- In the predictor variables independent of all the other predictor variables?
- Which predictor variables are the most important?
- Do the ranges of the predictor variables make sense?
- What are the distributions of the predictor variables?
- Remove outliers and keep outliers (does it have an effect of the final predictive model)?
- Remove 1%, 5%, and 10% of your data randomly and impute the values back using at least 3 imputation methods. How well did the methods recover the missing values? That is remove some data, check the % error on residuals for numeric data and check for bias and variance of the error.
- For categorical data, calculate the accuracy and a confusion matrix.

Abstract

Market Sales data contains customer specific information about a customer's details like age, income and shopping habits like amount spent, website visits etc. We aim to find if a customer will respond positively to a marketing campaign or not. Various techniques are used to explore data like correlation heatmaps and boxplots, Q-Q plots etc. Finally a logistic regression model is built to predict customer response. It appears that only some columns were of significance while predicting dependent variable. Different impute methods and their effectiveness is explored

About Dataset - Marketing Campaign

Context A response model can provide a significant boost to the efficiency of a marketing campaign by increasing responses or reducing expenses. The objective is to predict who will respond to an offer for a product or service

Content

1. AcceptedCmp1 - 1 if customer accepted the offer in the 1st campaign, 0 otherwise
2. AcceptedCmp2 - 1 if customer accepted the offer in the 2nd campaign, 0 otherwise
3. AcceptedCmp3 - 1 if customer accepted the offer in the 3rd campaign, 0 otherwise
4. AcceptedCmp4 - 1 if customer accepted the offer in the 4th campaign, 0 otherwise
5. AcceptedCmp5 - 1 if customer accepted the offer in the 5th campaign, 0 otherwise
6. Response (target) - 1 if customer accepted the offer in the last campaign, 0 otherwise
7. Complain - 1 if customer complained in the last 2 years
8. DtCustomer - date of customer's enrolment with the company
9. Education - customer's level of education
10. Marital - customer's marital status
11. Kidhome - number of small children in customer's household
12. Teenhome - number of teenagers in customer's household
13. Income - customer's yearly household income
14. MntFishProducts - amount spent on fish products in the last 2 years
15. MntMeatProducts - amount spent on meat products in the last 2 years
16. MntFruits - amount spent on fruits products in the last 2 years
17. MntSweetProducts - amount spent on sweet products in the last 2 years
18. MntWines - amount spent on wine products in the last 2 years
19. MntGoldProds - amount spent on gold products in the last 2 years
20. NumDealsPurchases - number of purchases made with discount

- 21. NumCatalogPurchases - number of purchases made using catalogue
- 22. NumStorePurchases - number of purchases made directly in stores
- 23. NumWebPurchases - number of purchases made through company's web site
- 24. NumWebVisitsMonth - number of visits to company's web site in the last month
- 25. Recency - number of days since the last purchase

[] ↪ 2 cells hidden

▸ Data Types

[] ↪ 7 cells hidden

▸ Null Values

[] ↪ 5 cells hidden

▸ Numeric Data Distribution

[] ↪ 8 cells hidden

▸ Data Transformation

[] ↪ 19 cells hidden

▼ Data Normalization

```
1 plt.figure(figsize=(20,7))
2 x = sns.boxplot(data=data)
3 x.set_xticklabels(x.get_xticklabels(),rotation=45)
```

```
[Text(0, 0, 'Age'),
Text(0, 0, 'Education'),
Text(0, 0, 'Income'),
Text(0, 0, 'Dt_Customer'),
Text(0, 0, 'Recency'),
Text(0, 0, 'NumDealsPurchases'),
```

Box plot shows Income having outliers and there is a vast difference between range of values between Income and other columns. We need to normalize data so all variables have equal weightage

```
Text(0, 0, 'AmountSpent')\n
1 from sklearn import preprocessing
2
3 # Create x to store scaled values as floats
4 x = data[["Age", "Education", "Income", "Dt_Customer", "Recency", "NumDealsPurchases", "NumWebVisitsMonth", "Complain", "Response", "Children", "AmountSpent"]]
5
6 # Preparing for normalizing
7 min_max_scaler = preprocessing.MinMaxScaler()
8
9 # Transform the data to fit minmax processor
10 x_scaled = min_max_scaler.fit_transform(x)
11
12 # Run the normalizer on the dataframe
13 data[["Age", "Education", "Income", "Dt_Customer", "Recency", "NumDealsPurchases", "NumWebVisitsMonth", "Complain", "Response", "Children", "AmountSpent"]] = x_scaled
14
15 data.head()
```

	Age	Education	Income	Dt_Customer	Recency	NumDealsPurchases	NumWebVisitsMonth	Complain	Response	Children	AmountSpent
0	0.378641	0.333333	0.084832	0.948498	0.585859	0.200000	0.35	0	1	0.000000	0.000000
1	0.407767	0.333333	0.067095	0.161660	0.383838	0.133333	0.25	0	0	0.666667	0.000000
2	0.300971	0.333333	0.105097	0.446352	0.262626	0.066667	0.20	0	0	0.000000	0.000000
3	0.116505	0.333333	0.037471	0.198856	0.262626	0.133333	0.30	0	0	0.333333	0.000000
4	0.145631	1.000000	0.085065	0.230329	0.949495	0.333333	0.25	0	0	0.333333	0.000000



```
1 plt.figure(figsize=(20,7))
2 x = sns.boxplot(data=data)
3 x.set_xticklabels(x.get_xticklabels(),rotation=45)
```

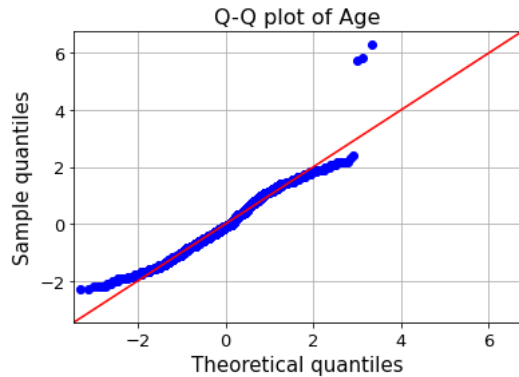
```

[Text(0, 0, 'Age'),
 Text(0, 0, 'Education'),
 Text(0, 0, 'Income'),
 Text(0, 0, 'Dt_Customer'),
 Text(0, 0, 'Recency'),
 Text(0, 0, 'NumDealsPurchases'),
 Text(0, 0, 'NumWebVisitsMonth'),
 Text(0, 0, 'Complain'),
 Text(0, 0, 'Response'),
 Text(0, 0, 'Children')]

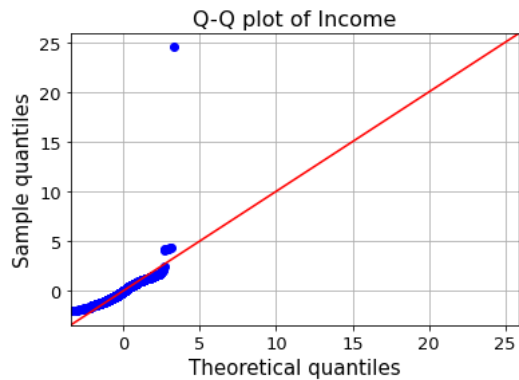
1 #checking the distribution of independent variables
2 data = data.dropna()
3 from statsmodels.graphics.gofplots import qqplot
4 data_norm=data[['Age', 'Income', 'Dt_Customer', 'Recency', 'NumWebVisitsMonth', 'AmountSpent']]
5 for c in data_norm.columns[:]:
6     plt.figure(figsize=(8,5))
7     fig=qqplot(data_norm[c],line='45',fit='True')
8     plt.xticks(fontsize=13)
9     plt.yticks(fontsize=13)
10    plt.xlabel("Theoretical quantiles",fontsize=15)
11    plt.ylabel("Sample quantiles",fontsize=15)
12    plt.title("Q-Q plot of {}".format(c),fontsize=16)
13    plt.grid(True)
14    plt.show()

```

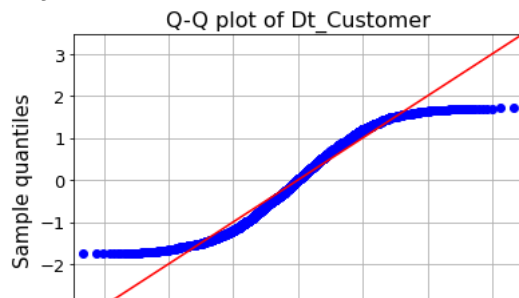
<Figure size 576x360 with 0 Axes>



<Figure size 576x360 with 0 Axes>



<Figure size 576x360 with 0 Axes>



- Q-Q plots show most of the data is normally distributed
- There are few Outliers in Income and age

<Figure size 576x360 with 0 Axes>

```
1 #pair plot to check the colinearity
2 # sns.pairplot(data)
```

```
1 #Using OLS for finding the p value to check the significant features
2 import statsmodels.api as sm
3
4 model = sm.OLS(data['Response'], data[["Age", "Education", "Income", "Dt_Customer",
5 "Recency", "NumDealsPurchases", "NumWebVisitsMonth",
6 "Complain", "Children", "AmountSpent", "NumPurchased",
7 "Prev_campaigns", "Marital_Status_Couple", "Marital_Status_Single"]]).fit()
8
9 # Print out the statistics
10 model.summary()
```

OLS Regression Results

Dep. Variable:	Response	R-squared:	0.301
Model:	OLS	Adj. R-squared:	0.297
Method:	Least Squares	F-statistic:	73.64
Date:	Sun, 05 Feb 2023	Prob (F-statistic):	3.34e-162
Time:	04:47:01	Log-Likelihood:	-465.46
No. Observations:	2240	AIC:	958.9
Df Residuals:	2226	BIC:	1039.
Df Model:	13		

Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
Age	-0.0404	0.057	-0.712	0.477	-0.152	0.071
Education	0.1118	0.023	4.859	0.000	0.067	0.157
Income	-0.0596	0.246	-0.242	0.809	-0.543	0.423
Dt_Customer	0.2200	0.025	8.872	0.000	0.171	0.269
Recency	-0.2388	0.022	-11.030	0.000	-0.281	-0.196
NumDealsPurchases	0.1661	0.063	2.648	0.008	0.043	0.289
NumWebVisitsMonth	0.1149	0.075	1.534	0.125	-0.032	0.262
Complain	0.0383	0.066	0.582	0.560	-0.091	0.167
Children	-0.1185	0.034	-3.469	0.001	-0.186	-0.052
AmountSpent	0.2087	0.057	3.642	0.000	0.096	0.321
NumPurchased	-0.1920	0.054	-3.571	0.000	-0.297	-0.087

- Education, Dt_Customer, Recency, Children, Amount spent, NumPurchases, Prev_campaign have a p value of 0.00 denoting a very high significance
- Income have a p value of 0.8 which shows it does not have a lot of significance.
- Marital_Status_Couple has p value 0.666 whereas Marital_Status_Single has 0.001 which is interesting as they both originate from same data column. It shows the significance of creating dummy variables for categorical data

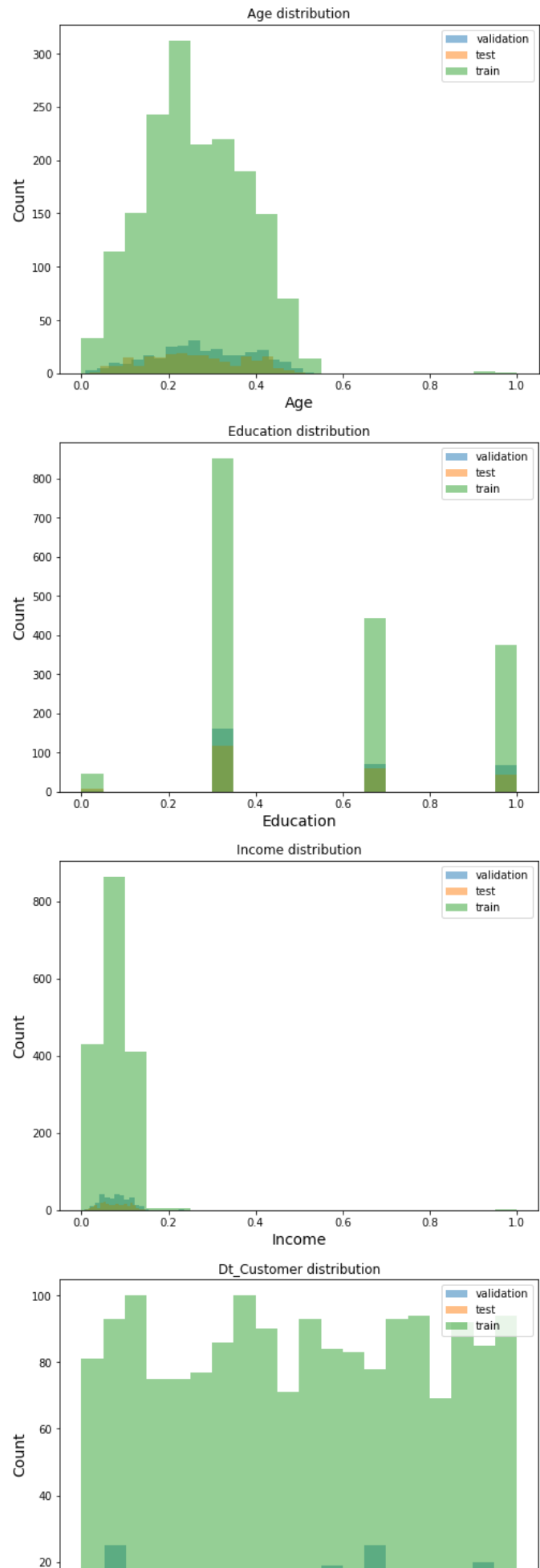
Logistic Regression

```
1 from sklearn.model_selection import train_test_split
2
3 X = data[ ["Age", "Education", "Income", "Dt_Customer",
4           "Recency", "NumDealsPurchases", "NumWebVisitsMonth",
5           "Complain", "Children", "AmountSpent", "NumPurchased",
6           "Prev_campaigns", "Marital_Status_Couple", "Marital_Status_Single"]]
7
8 y = data['Response']
9
10 #Splitting data into Training 76.5%, Validation set 13.5% and Test set 10%
11
12 X_t, X_test, y_t, y_test = train_test_split(X, y, test_size=0.1, random_state=1)
13
14 X_train, X_val, y_train, y_val = train_test_split(X_t, y_t, test_size=0.15, random_state=1)
```

Splitting data for train test and validation 10% testing 15% of remaining train data for validation

```
1 # Looking the data for test, training and validation set
2 X_test_plot = X_test[["Age", "Education", "Income", "Dt_Customer",
3                       "Recency", "NumDealsPurchases", "NumWebVisitsMonth",
4                       "Complain", "Children", "AmountSpent", "NumPurchased",
5                       "Prev_campaigns", "Marital_Status_Couple", "Marital_Status_Single"]]
6
7 X_val_plot = X_val[["Age", "Education", "Income", "Dt_Customer",
8                     "Recency", "NumDealsPurchases", "NumWebVisitsMonth",
9                     "Complain", "Children", "AmountSpent", "NumPurchased",
10                     "Prev_campaigns", "Marital_Status_Couple", "Marital_Status_Single"]]
11
12 X_train_plot = X_train[["Age", "Education", "Income", "Dt_Customer",
13                          "Recency", "NumDealsPurchases", "NumWebVisitsMonth",
14                          "Complain", "Children", "AmountSpent", "NumPurchased",
15                          "Prev_campaigns", "Marital_Status_Couple", "Marital_Status_Single"]]
16
17 # Plotting the data to see the histogram
18 for c in X_test_plot.columns[:]:
19     plt.figure(figsize=(8,6))
20     plt.hist(X_val_plot[c], bins=20, alpha=0.5, label="validation")
```

```
21 plt.hist(X_test_plot[c], bins=20, alpha=0.5, label="test")
22 plt.hist(X_train_plot[c], bins=20, alpha=0.5, label="train")
23 plt.xlabel(c, size=14)
24 plt.ylabel("Count", size=14)
25 plt.legend(loc='upper right')
26 plt.title("{} distribution".format(c))
27 plt.show()
```



Distribution shows our data splits are evenly distributed

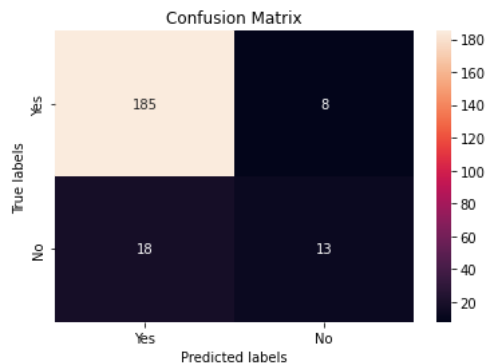
```
1 from sklearn import datasets, linear_model
2 from sklearn.metrics import mean_squared_error, r2_score
3 from sklearn.model_selection import train_test_split
4 import statsmodels.api as sm
5 from scipy import stats
6 import seaborn as sns
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.metrics import classification_report, confusion_matrix
9 from sklearn import metrics
10 from sklearn.preprocessing import LabelEncoder
11 from sklearn import svm
12 from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
13 from sklearn.inspection import permutation_importance
14 logreg=LogisticRegression()
15 logreg_final=logreg.fit(X_train,y_train)
16 print(logreg_final.score(X_train,y_train))
```

0.8832457676590777

▼ Making Predictions

Our model has 88.32% training accuracy

```
1 y_pred=logreg.predict(X_test)
2
3 cm=confusion_matrix(y_test, y_pred)#confusion matrix for the logistic model prediction
4
5 ax= plt.subplot()
6 sns.heatmap(cm, annot=True, fmt='g', ax=ax); #annot=True to annotate cells, fmt='g' to disable scientific notation
7
8 # labels, title and ticks
9 ax.set_xlabel('Predicted labels');
10 ax.set_ylabel('True labels');
11 ax.set_title('Confusion Matrix');
12 ax.xaxis.set_ticklabels(['Yes', 'No']);
13 ax.yaxis.set_ticklabels(['Yes', 'No']);
```



Above confusion matrix shows a good percentage of testing data is accurately predicted

```
1 print(classification_report(y_test, y_pred))
2 #classification report for logistic model prediction
```

	precision	recall	f1-score	support
0	0.91	0.96	0.93	193
1	0.62	0.42	0.50	31
accuracy			0.88	224
macro avg	0.77	0.69	0.72	224
weighted avg	0.87	0.88	0.87	224

We have a higher precision for "No" i.e 0 of 0.91 while precision for "Yes" is 0.62 indicating we have a more accurate prediction chance for a negative customer response

```
400 |
1 #Understanding the important features
2 import eli5
3 from eli5.sklearn import PermutationImportance
4 perm = PermutationImportance(logreg, random_state=1).fit(X_test, y_test)
5 eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

Weight	Feature
0.0384 ± 0.0134	Prev_campaigns
0.0098 ± 0.0067	NumWebVisitsMonth
0.0036 ± 0.0143	NumPurchased
0.0036 ± 0.0067	NumDealsPurchases
0 ± 0.0000	Complain
0 ± 0.0000	Income
-0.0027 ± 0.0044	Age
-0.0027 ± 0.0091	Children
-0.0036 ± 0.0261	Dt_Customer
-0.0045 ± 0.0253	Education
-0.0054 ± 0.0242	Recency
-0.0089 ± 0.0056	AmountSpent
-0.0107 ± 0.0134	Marital_Status_Single
-0.0152 ± 0.0134	Marital_Status_Couple

Features in increasing order of significance as evident from permutaion importance

1. Prev_campaigns
2. NumWebVisitsMonth
3. NumPurchased
4. NumDealsPurchases
5. Education
6. Marital_Status_Couple
7. AmountSpent
8. Recency

▾ Removing Outliers

```
1 data.Income.quantile(0.999)
```

0.23812708290722912

```
1 data.drop(data[data['Income'] >= 0.2].index, inplace = True)
```

```
1 data.isnull().sum()
```

```
Age                0
Education          0
Income             0
Dt_Customer        0
Recency            0
NumDealsPurchases  0
NumWebVisitsMonth  0
Complain           0
Response           0
Children           0
AmountSpent        0
NumPurchased       0
Prev_campaigns     0
Marital_Status_Couple 0
Marital_Status_Single 0
dtype: int64
```

```
1 # Create x to store scaled values as floats
2 x = data[["Age", "Education", "Income", "Dt_Customer", "Recency", "NumDealsPurchases", "NumWebVisitsMonth", "
3
4 # Preparing for normalizing
5 min_max_scaler = preprocessing.MinMaxScaler()
6
```

```

7 # Transform the data to fit minmax processor
8 x_scaled = min_max_scaler.fit_transform(x)
9
10 # Run the normalizer on the dataframe
11 data[["Age", "Education", "Income", "Dt_Customer", "Recency", "NumDealsPurchases", "NumWebVisitsMonth", "Children", "AmountSpent", "NumPurchased", "Prev_campaigns", "Marital_Status_Couple", "Marital_Status_Single"]]
12
13 data.head()

```

	Age	Education	Income	Dt_Customer	Recency	NumDealsPurchases	NumWebVisitsMonth	Complain	Response	Children	AmountSpent	NumPurchased	Prev_campaigns	Marital_Status_Couple	Marital_Status_Single
0	0.378641	0.333333	0.503625	0.948498	0.585859	0.200000	0.35	0	1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.407767	0.333333	0.398325	0.161660	0.383838	0.133333	0.25	0	0	0.666667	0.666667	0.666667	0.666667	0.666667	0.666667
2	0.300971	0.333333	0.623933	0.446352	0.262626	0.066667	0.20	0	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.116505	0.333333	0.222456	0.198856	0.262626	0.133333	0.30	0	0	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333
4	0.145631	1.000000	0.505009	0.230329	0.949495	0.333333	0.25	0	0	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333



```

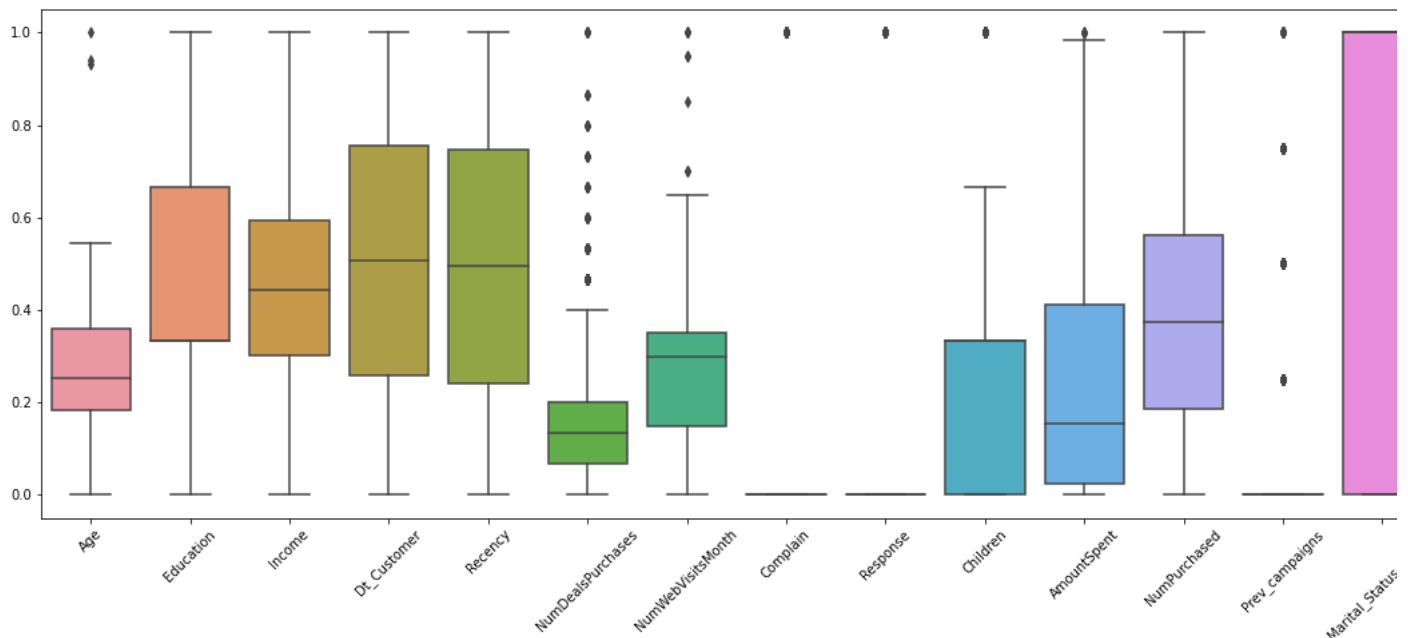
1 plt.figure(figsize=(20,7))
2 x = sns.boxplot(data=data)
3 x.set_xticklabels(x.get_xticklabels(),rotation=45)

```

```

[Text(0, 0, 'Age'),
Text(0, 0, 'Education'),
Text(0, 0, 'Income'),
Text(0, 0, 'Dt_Customer'),
Text(0, 0, 'Recency'),
Text(0, 0, 'NumDealsPurchases'),
Text(0, 0, 'NumWebVisitsMonth'),
Text(0, 0, 'Complain'),
Text(0, 0, 'Response'),
Text(0, 0, 'Children'),
Text(0, 0, 'AmountSpent'),
Text(0, 0, 'NumPurchased'),
Text(0, 0, 'Prev_campaigns'),
Text(0, 0, 'Marital_Status_Couple'),
Text(0, 0, 'Marital_Status_Single')]

```



We Normalize data again after removing outliers from Income column

```

1 from sklearn.linear_model import LogisticRegression
2
3 data = data.dropna()
4 X = data[["Age", "Education", "Income", "Dt_Customer",
5           "Recency", "NumDealsPurchases", "NumWebVisitsMonth",
6           "Complain", "Children", "AmountSpent", "NumPurchased",
7           "Prev_campaigns", "Marital_Status_Couple", "Marital_Status_Single"]]

```

```

8
9 y = data['Response']
10
11 #Splitting data into Training 76.5%, Validation set 13.5% and Test set 10%
12
13 X_t, X_test, y_t, y_test = train_test_split(X, y, test_size=0.1, random_state=1)
14
15 X_train, X_val, y_train, y_val = train_test_split(X_t, y_t, test_size=0.15, random_state=1)
16 logreg=LogisticRegression()
17 loggereg_final=logreg.fit(X_train,y_train)
18 print(loggereg_final.score(X_train,y_train))

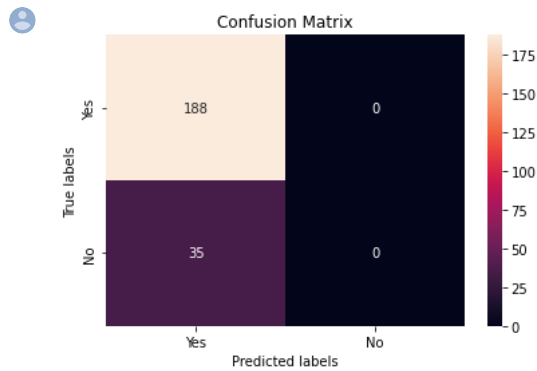
```

0.8518518518518519

```

1 y_pred=logreg.predict(X_test)
2
3 cm=confusion_matrix(y_test, y_pred)#confusion matrix for the logistic model prediction
4
5 ax= plt.subplot()
6 sns.heatmap(cm, annot=True, fmt='g', ax=ax); #annot=True to annotate cells, ftm='g' to disable scientific notation
7
8 # labels, title and ticks
9 ax.set_xlabel('Predicted labels');
10 ax.set_ylabel('True labels');
11 ax.set_title('Confusion Matrix');
12 ax.xaxis.set_ticklabels(['Yes', 'No']);
13 ax.yaxis.set_ticklabels(['Yes', 'No']);

```



```

1 print(classification_report(y_test, y_pred))
2 #classification report for logistic model prediction

```

	precision	recall	f1-score	support
0	0.84	1.00	0.91	188
1	0.00	0.00	0.00	35
accuracy			0.84	223
macro avg	0.42	0.50	0.46	223
weighted avg	0.71	0.84	0.77	223

```

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score
_warn_prf(average, modifier, msg_start, len(result))

```

```

1 from sklearn.impute import KNNImputer
2 from sklearn.preprocessing import MinMaxScaler
3 def create_missing(dataframe, percent, col):
4     dataframe.loc[dataframe.sample(frac = percent).index, col] = np.nan

```

```

1 data_original = data.copy()
2 create_missing(data, 0.01, 'Income')

```

```


1 #checking if the any data is missing
2 def checkMissing(dataset):
3     percent_missing = dataset.isnull().sum() * 100 / len(data)
4     null_values_total = dataset.isnull().sum()

```

```

5 missing_value_df = pd.DataFrame({
6     'Missing_Total' : null_values_total,
7     'percent_missing': percent_missing,
8 })
9 return missing_value_df
10
11 checkMissing(data)
12

```


	Missing_Total	percent_missing	
Age	0	0.000000	
Education	0	0.000000	
Income	22	0.988764	
Dt_Customer	0	0.000000	
Recency	0	0.000000	
NumDealsPurchases	0	0.000000	
NumWebVisitsMonth	0	0.000000	
Complain	0	0.000000	
Response	0	0.000000	
Children	0	0.000000	
AmountSpent	0	0.000000	
NumPurchased	0	0.000000	
Prev_campaigns	0	0.000000	
Marital_Status_Couple	0	0.000000	
Marital_Status_Single	0	0.000000	

▼ Average

```

1 number_1_idx = list(np.where(data['Income'].isna())[0])
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

	Missing_Total	percent_missing	
Age	0	0.0	
Education	0	0.0	
Income	0	0.0	
Dt_Customer	0	0.0	
Recency	0	0.0	
NumDealsPurchases	0	0.0	
NumWebVisitsMonth	0	0.0	
Complain	0	0.0	
Response	0	0.0	
Children	0	0.0	
AmountSpent	0	0.0	
NumPurchased	0	0.0	
Prev_campaigns	0	0.0	
Marital_Status_Couple	0	0.0	
Marital_Status_Single	0	0.0	

```

1 data_mn = data.iloc[number_1_idx]
2 data_og = data_original.iloc[number_1_idx]

```

```

1 # The mean squared error
2 print('Mean squared error: %.2f'% mean_squared_error(data_og['Income'], data_mn['Income']))
3 # The coefficient of determination: 1 is perfect prediction
4 print('Coefficient of determination: %.2f'% r2_score(data_og['Income'], data_mn['Income']))
5 r2 = r2_score(data_og['Income'], data_mn['Income'])
6 print('R^2 score on test set =',r2)
7

```


Mean squared error: 0.03
 Coefficient of determination: -0.00
 R^2 score on test set = -0.0009562959256552706

▼ Categorical mean

```

1 data = data_original.copy()
2 create_missing(data, 0.05, 'Income')
3 checkMissing(data)
4

```

	Missing_Total	percent_missing	
Age	0	0.000000	
Education	0	0.000000	
Income	111	4.988764	
Dt_Customer	0	0.000000	
Recency	0	0.000000	
NumDealsPurchases	0	0.000000	
NumWebVisitsMonth	0	0.000000	
Complain	0	0.000000	
Response	0	0.000000	
Children	0	0.000000	
AmountSpent	0	0.000000	
NumPurchased	0	0.000000	
Prev_campaigns	0	0.000000	
Marital_Status_Couple	0	0.000000	
Marital_Status_Single	0	0.000000	

```

1 number_5_idx = list(np.where(data['Income'].isna())[0])
2 data['Income'].fillna(data.groupby('Education')['Income'].transform('mean'), inplace = True)
3 checkMissing(data)

```

	Missing_Total	percent_missing
Age	0	0.0
Education	0	0.0



```

1 data_mn = data.iloc[number_5_idx]
2 data_og = data_original.iloc[number_5_idx]
3 # The mean squared error
4 print('Mean squared error: %.2f'% mean_squared_error(data_original['Income'], data['Income']))
5 # The coefficient of determination: 1 is perfect prediction
6 print('Coefficient of determination: %.2f'% r2_score(data_original['Income'], data['Income']))
7 r2 = r2_score(data_original['Income'], data['Income'])
8 print('R^2 score on test set =',r2)

```

Mean squared error: 0.00
Coefficient of determination: 0.95
R^2 score on test set = 0.9531689754402813

Children	0	0.0
----------	---	-----

▼ KNN impute

```

1 data = data_original.copy()
2 create_missing(data, 0.1, 'Income')
3 checkMissing(data)
4

```

	Missing_Total	percent_missing
Age	0	0.000000
Education	0	0.000000
Income	222	9.977528
Dt_Customer	0	0.000000
Recency	0	0.000000
NumDealsPurchases	0	0.000000
NumWebVisitsMonth	0	0.000000
Complain	0	0.000000
Response	0	0.000000
Children	0	0.000000
AmountSpent	0	0.000000
NumPurchased	0	0.000000
Prev_campaigns	0	0.000000
Marital_Status_Couple	0	0.000000
Marital_Status_Single	0	0.000000




```
1 number_10_idx = list(np.where(data['Income'].isna())[0])
```

```

1 imputer = KNNImputer(n_neighbors=5)
2 data = pd.DataFrame(imputer.fit_transform(data), columns = data.columns)
3 checkMissing(data)

```

	Missing_Total	percent_missing	
Age	0	0.0	
Education	0	0.0	
Income	0	0.0	
Dt_Customer	0	0.0	
Recency	0	0.0	
NumDealsPurchases	0	0.0	

```

1 data_mn = data.iloc[number_10_idx]
2 data Og = data_original.iloc[number_10_idx]
3 # The mean squared error
4 print('Mean squared error: %.2f'% mean_squared_error(data_original['Income'], data['Income']))
5 # The coefficient of determination: 1 is perfect prediction
6 print('Coefficient of determination: %.2f'% r2_score(data_original['Income'], data['Income']))
7 r2 = r2_score(data_original['Income'], data['Income'])
8 print('R^2 score on test set =',r2)

```

```

Mean squared error: 0.00
Coefficient of determination: 0.98
R^2 score on test set = 0.9794301329289421

```

Answer the following questions:

- Do the training and test sets have the same data?

No they donot have the same values

- In the predictor variables independent of all the other predictor variables?

There are some variables like amount spent and number of orders that are correlated

- Which predictor variables are the most important?

- Prev_campaigns
- NumWebVisitsMonth
- NumPurchased
- NumDealsPurchases
- Education
- Marital_Status_Couple
- AmountSpent
- Recency

- Remove outliers and keep outliers (does if have an effect of the final predictive model)?

From the confusion matrix it is evident we are getting better model and predictions

- Remove 1%, 5%, and 10% of your data randomly and impute the values back using at least 3 imputation methods. How well did the methods recover the missing values? That is remove some data, check the % error on residuals for numeric data and check for bias and variance of the error.

For our dataset while removing values for Income column all 3 methods have similar results although one can argue

All code in this note is available as open source through the MIT license.

All text and images are free to use under the Creative Commons Attribution 3.0 license. <https://creativecommons.org/licenses/by/3.0/us/>

These licenses let others distribute, remix, tweak, and build upon the work, even commercially, as long as they give credit for the original creation.

Copyright 2023 Makarand Madhavi