Pre-Processing Data in Python

Summary

- Missing Values.
- Data Format.
- Data Normalization.
- Grouping into Classes.
- Converting Categorical Variables to Numeric Variables.

Data Pre-Processing

- It is the process of converting or mapping data from a raw format to a more manageable format for later analysis.
- Also known as “Data Cleaning”.

Missing Values

- A data entry that is left empty in the data set.
- It can be reflected with "?", Zero or a blank cell.

Customer	Customer Type	Payment Type	Purchases	Sales	Refunds	Country	Continent
10000	Person	Cash	120000.0	150000.0	240	Canada	America
10001	Company	Cash	NaN	651750.0	1043	Japan	Asia
10002	Company	Credit Card	451000.0	563750.0	902	Mexico	America
10003	Company	Transfer	565000.0	706250.0	1130	Spain	Europe
10004	Person	Transfer	512300.0	NaN	1024	Argentina	America

Missing Values

- Common Strategies
  - Review source data and fill in missing values.
  - Remove missing values.
    - Delete the entire variable.
    - Delete the entry with the missing data.
  - Replace missing values.
    - Replace with the average of the entire variable.
    - Replace with mode if categorical variable.
    - Replace based on collected data.
  - Stay with missing values.

Missing Values in Python

- Removing missing values in Python.

- dropna()
  - axis = 0
    Delete whole row
  - axis = 1
    Remove entire column

Customer	Customer Type	Payment Type	Purchases	Sales	Refunds	Country	Continent
10000	Person	Cash	120000.0	150000.0	240	Canada	America
10001	Company	Cash	NaN	651750.0	1043	Japan	Asia
10002	Company	Credit Card	451000.0	563750.0	902	Mexico	America
10003	Company	Transfer	565000.0	706250.0	1130	Spain	Europe
10004	Person	Transfer	512300.0	NaN	1024	Argentina	America

```
df_test.dropna(subset=["Purchases", "Sales"], axis=0, inplace = True)
df_test.head()
```

Customer	Customer Type	Payment Type	Purchases	Sales	Refunds	Country	Continent
10000	Person	Cash	120000.0	150000.0	240	Canada	America
10002	Company	Credit Card	451000.0	563750.0	902	Mexico	America
10003	Company	Transfer	565000.0	706250.0	1130	Spain	Europe
10005	Person	Transfer	415500.0	519375.0	0	Canada	America
10006	Company	Credit Card	696300.0	870375.0	1392	EEUU	America

Missing Values in Python

- Replacing missing values in Python.

- replace()

```
my_mean = df_test["Sales"].mean()
df_test["Sales"].replace(np.nan, int(my_mean), inplace = True)
df_test.head()
```

Customer	Customer Type	Payment Type	Purchases	Sales
10000	Person	Cash	120000.0	150000.0
10002	Company	Credit Card	451000.0	563750.0
10003	Company	Transfer	565000.0	706250.0
10004	Person	Transfer	512300.0	NaN
10005	Person	Transfer	415500.0	519375.0

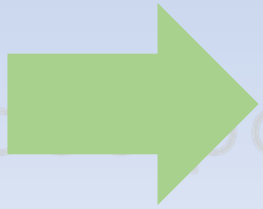
Customer	Customer Type	Payment Type	Purchases	Sales
10000	Person	Cash	120000.0	150000.0
10002	Company	Credit Card	451000.0	563750.0
10003	Company	Transfer	565000.0	706250.0
10004	Person	Transfer	512300.0	553509.0
10005	Person	Transfer	415500.0	519375.0

Data Format

- Different origins usually lead to different formats.

- A common data format makes data easy to compare and interpret.

Gender
0
1
Man
Woman
Male
Female
M
F



Gender
Male
Female
Male
Female
Male
Female
Male
Female

Data Format

- Converting numeric variables.

```
df_test["Purchases"] = df_test["Purchases"]/1000
df_test["Sales"] = df_test["Sales"]/1000
df_test.rename(columns={"Purchases":"Purchases in in thousands", "Sales":"Sales in thousands"}, inplace = True)
df_test.head()
```

Customer	Customer Type	Payment Type	Purchases	Sales
10000	Person	Cash	120000.0	150000.0
10002	Company	Credit Card	451000.0	563750.0
10003	Company	Transfer	565000.0	706250.0
10004	Person	Transfer	512300.0	553509.0
10005	Person	Transfer	415500.0	519375.0

Customer	Customer Type	Payment Type	Purchases in in thousands	Sales in thousands
10000	Person	Cash	120.0	150.000
10002	Company	Credit Card	451.0	563.750
10003	Company	Transfer	565.0	706.250
10004	Person	Transfer	512.3	553.509
10005	Person	Transfer	415.5	519.375

Data Format

- Correcting data type.

- dtypes()

```
df_test.dtypes
```

```
Customer          int64
Customer Type    object
Payment Type     object
Purchases       float64
Sales            object
Refunds           int64
Country          object
Continent        object
```

- astype()

```
df_test = df_test.astype({'Sales':'float'})
```

```
df_test.dtypes
```

```
Customer          object
Customer Type    object
Payment Type     object
Purchases       float64
Sales            float64
Refunds           int64
Country          object
Continent        object
```

Normalization

- It consists of putting the data in a similar range to be able to compare them.

Employee ID	First Name	Last Name	Age	Worked years	Salary	Status	Grade
1000001	John	Denver	23	1	500	Single	Elementary
1000002	Peter	Hank	30	3	900	Married	High School
1000003	Jack	Sullivan	27	2	900	Married	High School
1000004	Marco	Aurelio	40	8	1500	Married	Master Degree
1000005	Claudia	Perez	35	5	1300	Single	Master Degree
1000006	Sally	Royal	19	1	1400	Single	Graduate
1000007	Peter	Miller	33	4	600	Married	Graduate
1000008	Susan	Gordon	35	10	2000	Married	Master Degree

Normalization

- Data Normalization Methods.

1) Simple Feature Scaling. $$X_{new} = \frac{X_{current}}{X_{maximum}} \qquad 0 <= X_{new} <= 1$$

2) Min-Max. $$X_{new} = \frac{X_{current} - X_{minimum}}{X_{maximum} - X_{minimum}} \qquad 0 <= X_{new} <= 1$$

3) Z-score. $$X_{new} = \frac{X_{current} - Mean}{Standard\ Deviation} \qquad -3 <= X_{new} <= 3$$

https://en.wikipedia.org/wiki/Normal_distribution

Normalization

- Applying Simple Feature Scaling method in Python.

```
df_employees[['Age','Worked years','Salary']].head()
```

```
df_norm1["Age"] = df_norm1["Age"] / df_norm1["Age"].max()
df_norm1["Worked years"] = df_norm1["Worked years"] / df_norm1["Worked years"].max()
df_norm1["Salary"] = df_norm1["Salary"] / df_norm1["Salary"].max()
```

Age	Worked years	Salary
23	1	500
30	3	900
27	2	900
40	8	1500
35	5	1300

Age	Worked years	Salary
0.575	0.1	0.25
0.750	0.3	0.45
0.675	0.2	0.45
1.000	0.8	0.75
0.875	0.5	0.65

Normalization

- Applying Min-Max method in Python.

```
df_employees[['Age','Worked years','Salary']].head()
```

```
df_norm2 = df_employees

df_norm2["Age"] = (df_norm2["Age"] - df_norm2["Age"].min()) / (df_norm2["Age"].max() - df_norm2["Age"].min())
df_norm2["Worked years"] = (df_norm2["Worked years"] - df_norm2["Worked years"].min()) / (df_norm2["Worked years"].max() - df_norm2["Worked years"].min())
df_norm2["Salary"] = (df_norm2["Salary"] - df_norm2["Salary"].min()) / (df_norm2["Salary"].max() - df_norm2["Salary"].min())
```

Age	Worked years	Salary
23	1	500
30	3	900
27	2	900
40	8	1500
35	5	1300

Age	Worked years	Salary
0.190476	0.000000	0.000000
0.523810	0.222222	0.266667
0.380952	0.111111	0.266667
1.000000	0.777778	0.666667
0.761905	0.444444	0.533333

Normalization

- Applying Z-score method in Python.

```
df_employees[['Age','Worked years','Salary']].head()
```

```
df_norm3 = df_employees

df_norm3["Age"] = (df_norm3["Age"] - df_norm3["Age"].mean()) / df_norm3["Age"].std()
df_norm3["Worked years"] = (df_norm3["Worked years"] - df_norm3["Worked years"].mean()) / df_norm3["Worked years"].std()
df_norm3["Salary"] = (df_norm3["Salary"] - df_norm3["Salary"].mean()) / df_norm3["Salary"].std()
```

Age	Worked years	Salary
23	1	500
30	3	900
27	2	900
40	8	1500
35	5	1300

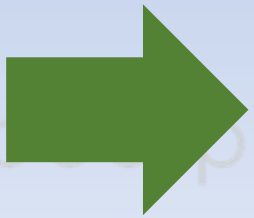
Age	Worked years	Salary
-1.044119	-0.989598	-1.264654
-0.036004	-0.380615	-0.471146
-0.468053	-0.685106	-0.471146
1.404160	1.141844	0.719117
0.684078	0.228369	0.322363

Grouping into Classes

- Grouping the data into classes.

- Sales have a range that goes from 100,000 to a little more than 900,000.

Sales
150000.0
563750.0
706250.0
553509.0
519375.0
870375.0
926250.0
676250.0
103750.0
567925.0



Classes	Class Range
Low	0 – 299,999
Medium	300,000 – 599,999
High	600,000 – 999,999

https://en.wikipedia.org/wiki/Frequency_distribution

Grouping into Classes

- Grouping with Python.

```
my_class = np.linspace(min(df_test["Sales"]), max(df_test["Sales"]), 4)
group_names = ["Low","Medium","High"]
df_test["Sales Category"] = pd.cut(df_test["Sales"], my_class, labels = group_names, include_lowest = True)
```

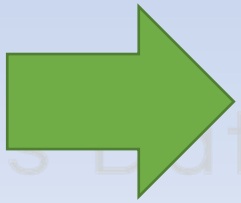
Payment Type	Purchases	Sales	Refunds	Country	Continent	Sales Category
Cash	120000.0	150000.0	240	Canada	America	Low
Credit Card	451000.0	563750.0	902	Mexico	America	Medium
Transfer	565000.0	706250.0	1130	Spain	Europe	High
Transfer	512300.0	553509.0	1024	Argentina	America	Medium
Transfer	415500.0	519375.0	0	Canada	America	Medium

Variable Conversion

- Converting categorical variables to numeric variables.
- get_dummies()

```
pd.get_dummies(df_test['Payment Type'])
```

Customer	Customer Type	Payment Type
10000	Person	Cash
10002	Company	Credit Card
10003	Company	Transfer
10004	Person	Transfer
10005	Person	Transfer
10006	Company	Credit Card
10007	Person	Cash
10008	Company	Cash
10009	Company	Cash
10010	Company	Cash



One-hot Encoding

Cash	Credit Card	Transfer
1	0	0
0	1	0
0	0	1
0	0	1
0	0	1
0	1	0
1	0	0
1	0	0
1	0	0
1	0	0