وزارة التعليم العالي

جامعة حمص

الكلية التطبيقية

قسم تقنيات حاسوب

السنة الثالثة

قواعد معطيات٣

القسم العملي

المحاضرة الثالثة

اعداد المدرسين:

م. زينب مراد **م. بتول الليوس**

Preprocessing using python

Data Cleaning:

I. Handling Missing Data.

1. Ignore tuples: usually done when class label is missing.
2. Eliminate attributes that having missing values.
3. Fill in the missing value manually.
4. Estimate missing values.

Cars Example:

Step 1: import required libraries:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.compose import ColumnTransformer
from scipy.stats import chi2_contingency
import matplotlib.pyplot as plt
```

Step 2: Load Dataset with Missing Values:

```
missing_c_df = pd.read_csv('car_pre.csv')
missing_c_df
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
...	...	...	...	...	...
995	Toyota	Black	35820.0	4.0	32042.0
996	NaN	White	155144.0	3.0	5716.0
997	Nissan	Blue	66604.0	4.0	31570.0
998	Honda	White	215883.0	4.0	4001.0
999	Toyota	Blue	248360.0	4.0	12732.0

Step 3: Fill in the missing value manually:

- **fillna() function:** **#use fillna() function with just passing the value to replace NaN:**

```
missing_c_df['Odometer (KM)'].fillna(missing_c_df['Odometer (KM)'].mean())
```

```
0       35431.0
1      192714.0
2       84714.0
3      154365.0
4      181577.0
         ...
995     35820.0
996    155144.0
997     66604.0
998    215883.0
999    248360.0
Name: Odometer (KM), Length: 1000, dtype: float64
```

 # view first 10 lines using head() function:

```
missing_c_df.head(10)
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
5	Honda	Red	42652.0	4.0	23883.0
6	Toyota	Blue	163453.0	4.0	8473.0
7	Honda	White	NaN	4.0	20306.0
8	NaN	White	130538.0	4.0	9374.0
9	Honda	Blue	51029.0	4.0	26683.0

 # get value counts for Make attribute:

```
missing_c_df['Make'].value_counts()
```

```
Make
Toyota    379
Honda     292
Nissan    183
BMW        97
Name: count, dtype: int64
```

 # filling Make attribute with most frequent value:

```
missing_c_df['Make'].fillna(missing_c_df['Make'].value_counts().index[0],inplace=True)
missing_c_df.head(10)
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.000000	4.0	15323.0
1	BMW	Blue	192714.000000	5.0	19943.0
2	Honda	White	84714.000000	4.0	28343.0
3	Toyota	White	154365.000000	4.0	13434.0
4	Nissan	Blue	181577.000000	3.0	14043.0
5	Honda	Red	42652.000000	4.0	23883.0
6	Toyota	Blue	163453.000000	4.0	8473.0
7	Honda	White	131253.237895	4.0	20306.0
8	Toyota	White	130538.000000	4.0	9374.0
9	Honda	Blue	51029.000000	4.0	26683.0

- **dropna() function:** **# drop records with missing Doors attribute:**

```
missing_c_df.dropna(axis=0, how='any', inplace=True, subset=['Doors'])
```

II. Handling Noisy Data: by using **data smoothing** techniques that remove noise from the data:
- Binning (which we'll focus on), Clustering, Regression,.....

Binning:

Binning is top-down splitting technique based on specified number of bins. To apply binning, we follow these steps:

1- The data values are first **sorted**.
2- Data values are partitioned into a number of buckets or bins by applying:
   - Equal-width (distance) partitioning: divides the range into N intervals of equal size.
     - ✓ If A and B are the lowest and highest values of the attribute, the width of intervals will be: W = (B-A)/N.
   - Equal-depth (frequency) partitioning: divides the range into N intervals, each containing approximately same number of samples.
3- smooth (replace) each bin value by **bin means**, **bin median**, or **bin boundaries**.

Pandas offers us two functions for binning:

I. cut() function applies equal-width partitioning, we specify the number of bins we need and Pandas do the work of calculating equal-sized bins for us.
II. qcut() (quantile cut) function applies equal-depth partitioning where the number of elements in each bin will roughly the same but different bins width.

To apply binning on Odometer (KM) attribute:

- We sort the values first.
- For cut() function we specify the number of bins.
- For qcut() instead of number of bins we must specify (q). for example, when q is equal to 5, we are telling pandas to cut the Odometer attribute into 5 equal quantiles, i.e. 0-20%, 20-40%, 40-60%, 60-80% and 80-100% buckets/bins.

1- sort the values using sort_values() function:

```
missing_c_df.sort_values('Odometer (KM)',inplace=True)
missing_c_df
```

	Make	Colour	Odometer (KM)	Doors	Price
891	Nissan	Black	10148.0	4.0	27337.0
719	Toyota	White	10217.0	4.0	22883.0
813	Toyota	White	10247.0	4.0	32566.0
415	Honda	White	10953.0	4.0	16636.0
403	Nissan	Blue	10954.0	4.0	30439.0
...	...	...	...	...	...
83	Toyota	Blue	248447.0	4.0	5708.0
988	Nissan	Black	248736.0	4.0	8358.0
893	Toyota	White	248815.0	4.0	9785.0
841	Honda	Blue	248899.0	4.0	5834.0
617	Nissan	Blue	249860.0	4.0	14524.0

2- **specify the number of bins using cut() function:**

```
pd.cut(missing_c_df['Odometer (KM)'],bins=5).head()
```

```
891    (9908.288, 58090.4]
719    (9908.288, 58090.4]
813    (9908.288, 58090.4]
415    (9908.288, 58090.4]
403    (9908.288, 58090.4]
Name: Odometer (KM), dtype: category
Categories (5, interval[float64]): [(9908.288, 58090.4] < (58090.4, 106032.8] < (106032.8, 153975.2] < (153975.2, 201917.6] < (201917.6, 249860.0]]
```

```
pd.cut(missing_c_df['Odometer (KM)'],bins=5, labels=['very short', 'short', 'med','long','very long'])
```

```
891     very short
719     very short
813     very short
415     very short
403     very short
           ...
83       very long
988      very long
893      very long
841      very long
617      very long
Name: Odometer (KM), Length: 950, dtype: category
Categories (5, object): [very short < short < med < long < very long]
```

```
missing_c_df['Odometer (KM)'] = pd.cut(missing_c_df['Odometer (KM)'],bins=5, labels=['very short', 'short', 'med','long','very long'])
```

```
missing_c_df
```

	Make	Colour	Odometer (KM)	Doors	Price
891	Nissan	Black	very short	4.0	27337.0
719	Toyota	White	very short	4.0	22883.0
813	Toyota	White	very short	4.0	32566.0
415	Honda	White	very short	4.0	16636.0
403	Nissan	Blue	very short	4.0	30439.0
...	...	...	...	...	...
83	Toyota	Blue	very long	4.0	5708.0
988	Nissan	Black	very long	4.0	8358.0
893	Toyota	White	very long	4.0	9785.0
841	Honda	Blue	very long	4.0	5834.0
617	Nissan	Blue	very long	4.0	14524.0

Or we must specify the q instead of number of bins using qcut() function:

```
pd.qcut(missing_c_df['Odometer (KM)'],q=5)
```

```
891    (10147.999, 59317.8]
719    (10147.999, 59317.8]
813    (10147.999, 59317.8]
415    (10147.999, 59317.8]
403    (10147.999, 59317.8]
              ...
83     (198935.0, 249860.0]
988    (198935.0, 249860.0]
893    (198935.0, 249860.0]
841    (198935.0, 249860.0]
617    (198935.0, 249860.0]
Name: Odometer (KM), Length: 950, dtype: category
Categories (5, interval[float64]): [(10147.999, 59317.8] < (59317.8, 111192.4] < (111192.4, 149164.0] < (149164.0, 198935.0] < (198935.0, 249860.0]]
```

Data Integration:

Careful integration of the data from multiple sources may help reduce/avoid redundancies and inconsistencies and improve the accuracy and speed of the subsequent data mining process. Redundant attributes can be detected by correlation tests:

- chi-square test for nominal data (Categorical Data).
- correlation coefficient and covariance for numeric attributes.

1- Calculate Chi-square test:

To calculate Chi-square test using Pandas and SciPy (Science Python which is a python module is used for statistics) Libraries we'll perform the following steps:

I. generate contingency table using crosstab function provided by Pandas for both attributes Odometer (KM) and Make.

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.compose import ColumnTransformer
from scipy.stats import chi2_contingency
import matplotlib.pyplot as plt
```

```
contin_table = pd.crosstab(missing_c_df['Odometer (KM)'],missing_c_df['Make'])
contin_table
```

Make	BMW	Honda	Nissan	Toyota
Odometer (KM)				
very short	21	66	31	67
short	16	49	24	87
med	22	64	44	100
long	20	47	40	72
very long	14	50	36	80

II. SciPy gives us useful function to calculate this test called chi2_contingency() which it takes contingency table and correction parameter. The function returns 4 values:

III. Correction: If True, *and* the degrees of freedom is 1, apply Yates' correction for continuity. The effect of the correction is to adjust each observed value by 0.5 towards the corresponding expected value.

#	Returned Value	Explanation
1	chi2	Test value.
2	dof	Degree of freedom.
3	p	P-value of the test (used for check test goodness).
4	expected	Expected values.

```
chi2, p, dof, expected = chi2_contingency(contin_table, correction=False)
```

```
print("Chi-Square result:",chi2)
print("P-Value:",p)
print("Degree of freedom",dof)
print("Expected Value:",expected)
```

```
Chi-Square result: 13.61080194837294
P-Value: 0.32624880996096917
Degree of freedom 12
Expected Value: [[18.11052632 53.74736842 34.07894737 79.06315789]
 [17.22947368 51.13263158 32.42105263 75.21684211]
 [22.51578947 66.82105263 42.36842105 98.29473684]
 [17.52315789 52.00421053 32.97368421 76.49894737]
 [17.62105263 52.29473684 33.15789474 76.92631579]]
```

2- Correlation Coefficient:

Correlation coefficient (also called Pearson s correlation coefficient) measures the linear association between two numeric attributes, A and B:

I. $r_{A,B} > 0$, A and B are positively correlated.
II. $r_{A,B} = 0$, independent and there is no correlation between them (A and B).
III. $r_{AB} < 0$, A and B are negatively correlated.

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.compose import ColumnTransformer
from scipy.stats import chi2_contingency
import matplotlib.pyplot as plt
```

```
car_df = pd.read_csv('car_sdd.csv')
car_df
```

	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	4000.0
1	Honda	Red	87899	4	5000.0
2	Toyota	Blue	32549	3	7000.0
3	BMW	Black	11179	5	22000.0
4	Nissan	White	213095	4	3500.0
5	Toyota	Green	99213	4	4500.0
6	Honda	Blue	45698	4	7500.0
7	Honda	Blue	54738	4	7000.0
8	Toyota	White	60000	4	6250.0
9	Nissan	White	31600	4	9700.0

```
corr_coeffs = np.corrcoef(car_df['Odometer (KM)'],car_df['Price'])
corr_coeffs
```

```
array([[ 1.        , -0.63178085],
       [-0.63178085,  1.        ]])
```

The result (negatively correlated): as car walked more its price will decrease and vice versa.

3- Covariance:

in python we calculate covariance between A and B as follows:

```
car_df['Odometer (KM)'].cov(car_df['Price'])
```

```
-210657447.77777776
```

Data Transformation:

Strategies for data transformation include the following:

1- Adding new attributes using given attributes:

Let's adding new attributes to our dataset. To do that in python we use lists or derived them using given attributes. The two new attributes are:

- ✓ Fuel economy which presents how many fuel litters is used by a car within 100KM.
- ✓ Total used Fuel which presents how many fuel litters are used by a car for total distance it walked (Odometer attribute).

Adding new attributes to our dataset:

```
fuel_per_100KM = [7.5,9.2,5.0,9.6,8.7,4.7,7.6,8.7,3.0,4.5]
car_df['Fuel per 100KM'] = fuel_per_100KM
car_df
```

	Make	Colour	Odometer (KM)	Doors	Price	Fuel per 100KM
0	Toyota	White	150043	4	4000.0	7.5
1	Honda	Red	87899	4	5000.0	9.2
2	Toyota	Blue	32549	3	7000.0	5.0
3	BMW	Black	11179	5	22000.0	9.6
4	Nissan	White	213095	4	3500.0	8.7
5	Toyota	Green	99213	4	4500.0	4.7
6	Honda	Blue	45698	4	7500.0	7.6
7	Honda	Blue	54738	4	7000.0	8.7
8	Toyota	White	60000	4	6250.0	3.0
9	Nissan	White	31600	4	9700.0	4.5

```
car_df['Total Fuel used'] = car_df['Odometer (KM)']/100 *car_df['Fuel per 100KM']
car_df
```

	Make	Colour	Odometer (KM)	Doors	Price	Fuel per 100KM	Total Fuel used
0	Toyota	White	150043	4	4000.0	7.5	11253.225
1	Honda	Red	87899	4	5000.0	9.2	8086.708
2	Toyota	Blue	32549	3	7000.0	5.0	1627.450
3	BMW	Black	11179	5	22000.0	9.6	1073.184
4	Nissan	White	213095	4	3500.0	8.7	18539.265
5	Toyota	Green	99213	4	4500.0	4.7	4663.011
6	Honda	Blue	45698	4	7500.0	7.6	3473.048
7	Honda	Blue	54738	4	7000.0	8.7	4762.206
8	Toyota	White	60000	4	6250.0	3.0	1800.000
9	Nissan	White	31600	4	9700.0	4.5	1422.000

2- Normalization:

To help avoid **dependence** on the choice of **measurement units**, the data should be **normalized** or **standardized** (give all attributes an equal weight).

This involves transforming the data to fall within a smaller or common range such as [-1,1] or [0,1]. For data normalization we have many methods:

Let's suppose we have Attribute A where its values are within range from $\boldsymbol{min_A}$ to $\boldsymbol{max_A}$ and an observation $\boldsymbol{v_i}$. The new value $\boldsymbol{v_i'}$ in new ranges ($\boldsymbol{new\_min_A}$ to $\boldsymbol{new\_max_A}$) will be:

- ✓ **Min-max** Normalization: (which we'll focus on)

$$v_i' = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

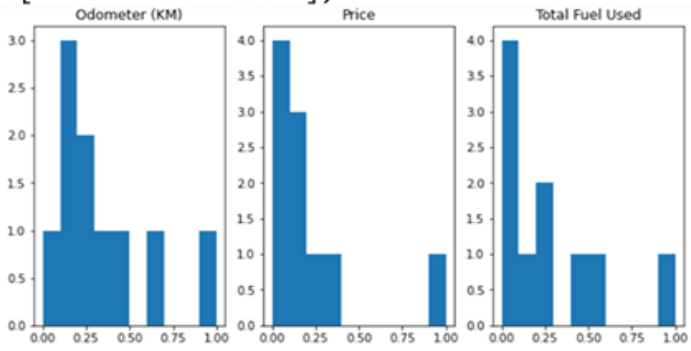
- ✓ **Z-score** Normalization.

$$v_i' = \frac{v_i - \bar{A}}{\sigma_A} \; where\ \bar{A}\ mean\ and\ \sigma_A\ std\ for\ attribute\ A$$

let's apply min-max normalization using Scikit Learn or (sklearn) for short.

I. sklearn provides a class called *MinMaxScaler* which helps us applying min-max normalization approach.
II. First, we take our numerical data by **dropping** all other **unnecessary** attributes then applying the transform using *fit_transform* method.
III. After that, we use matplotlib to draw histograms for all normalized attributes to ensure our normalization.

```
scaler = preprocessing.MinMaxScaler()
numeric_attribute = car_df.drop(columns=['Make','Colour','Doors','Fuel per 100KM'])
numeric_attribute = scaler.fit_transform(numeric_attribute)
#save transfered data in Pandas dataframe
numeric_attribute = pd.DataFrame(data=numeric_attribute,columns=['Odometer (KM)',
'Price','Total Fuel Used'])
#create multiple subplots in the same figure
fig, (ax1,ax2,ax3) = plt.subplots(nrows=1, ncols=3, figsize=(10,5))
ax1.set_title("Odometer (KM)")
ax1.hist(numeric_attribute['Odometer (KM)'])
ax2.set_title("Price")
ax2.hist(numeric_attribute['Price'])
ax3.set_title("Total Fuel Used")
ax3.hist(numeric_attribute['Total Fuel Used'])
```



3- Binarization:

Some data mining algorithms may need that both continuous and discrete attributes to be transformed into one or more **binary attributes**.

I. We specify **categorical** features like Doors, colors, …
II. We'll use sklearn class called *OneHotEncoder* which take a list of attributes.
III. To fit the encoding to the attributes, we use *ColumnTransformer* class:
- ✓ It takes a List of *(name, transformer, attributes)* tuples specifying the transformer objects to be applied to subsets of the data.
- ✓ Reminder parameter takes two possible values '*drop*' and '*passthrough*' to indicate to drop the attributes or to pass them through untransformed.

categorical_features = ['Make' , 'Colour' , 'Doors']

one_hot = preprocessing.OneHotEncoder()

transformer = ColumnTransformer([("one_hot",one_hot,categorical_features)] , remainder='passthrough')

transformed_data = pd.DataFrame(transformer.fit_transform(car_df))

transformed_data

and the result will look like:

```
categorical_features = ['Make','Colour','Doors']
```

```
one_hot = preprocessing.OneHotEncoder()
```

```
transformer = ColumnTransformer([("one_hot",one_hot,categorical_features)],remainder='passthrough')
```

```
transformed_data = pd.DataFrame(transformer.fit_transform(car_df))
```

```
transformed_data
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	150043.0	4000.0	7.5	11253.225
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	87899.0	5000.0	9.2	8086.708
2	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	32549.0	7000.0	5.0	1627.450
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	11179.0	22000.0	9.6	1073.184
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	213095.0	3500.0	8.7	18539.265
5	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	99213.0	4500.0	4.7	4663.011
6	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	45698.0	7500.0	7.6	3473.048
7	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	54738.0	7000.0	8.7	4762.206
8	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	60000.0	6250.0	3.0	1800.000
9	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	31600.0	9700.0	4.5	1422.000

#change attributes names to be clear to read:

dummies = car_df[["Make","Colour","Doors"]].astype(str)

dm = pd.get_dummies(dummies)

dm

_Honda	Make_Nissan	Make_Toyota	Colour_Black	Colour_Blue	Colour_Green	Colour_Red	Colour_White	Doors_3	Doors_4	[
0	0	1	0	0	0	0	1	0	1	
1	0	0	0	0	0	1	0	0	1	
0	0	1	0	1	0	0	0	1	0	
0	0	0	1	0	0	0	0	0	0	
0	1	0	0	0	0	0	1	0	1	
0	0	1	0	0	1	0	0	0	1	
1	0	0	0	1	0	0	0	0	1	
1	0	0	0	1	0	0	0	0	1	
0	0	1	0	0	0	0	1	0	1	
0	1	0	0	0	0	0	1	0	1	

and the result will look like:

```
dummies = car_df[["Make","Colour","Doors"]].astype(str)
```

```
dm = pd.get_dummies(dummies)
```

```
dm
```

	Make_BMW	Make_Honda	Make_Nissan	Make_Toyota	Colour_Black	Colour_Blue	Colour_Green	Colour_Red	Colour_White	Doors_3	Doors_4	Doors_5
0	False	False	False	True	False	False	False	False	True	False	True	False
1	False	True	False	False	False	False	False	True	False	False	True	False
2	False	False	False	True	False	True	False	False	False	True	False	False
3	True	False	False	False	True	False	False	False	False	False	False	True
4	False	False	True	False	False	False	False	False	True	False	True	False
5	False	False	False	True	False	False	True	False	False	False	True	False
6	False	True	False	False	False	True	False	False	False	False	True	False
7	False	True	False	False	False	True	False	False	False	False	True	False
8	False	False	False	True	False	False	False	False	True	False	True	False
9	False	False	True	False	False	False	False	False	True	False	True	False

Note: We can also consider Attribute construction and aggregation as data reduction techniques.

References:

1- Pandas docs [https://pandas.pydata.org/docs/].
2- NumPy docs [https://numpy.org/doc/].
3- Matplotlib docs [https://matplotlib.org/3.3.3/contents.html].
4- SciKit-learn docs [https://scikit-learn.org/0.21/documentation.html].
5- SciPy docs [https://docs.scipy.org/doc/].