# HW 2 Solutions to Worked Problems

David Gerbing
The School of Business
Portland State University
gerbing@pdx.edu

The following shows some useful data manipulations that are applicable to all data analysis: subsetting a data table, converting variable types, and variable transformations.

```
from datetime import datetime as dt
now = dt.now()
print ("Analysis on", now.strftime("%Y-%m-%d"), "at", now.strftime("%H:%M"))
```

```
    Analysis on 2020-07-10 at 11:23
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

## Data Wrangling, Pre-Processing I

*a. Read the data file.*

```
d = pd.read_csv('http://web.pdx.edu/~gerbing/data/Boston.csv')
```

*b. How many examples (rows of data) are there in the data file?*

```
d.shape
```

```
    (506, 15)
```

There are 506 rows of data.

*c. List the first 5 rows and the variable names.*

```
d.head()
```

| | Unnamed: 0 | crim | zn | indus | chas | nox | rm | age | dis | rad | tax | pt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.00632 | 18.0 | 2.31 | 0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1 | 296 | |
| **1** | 2 | 0.02731 | 0.0 | 7.07 | 0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 242 | |
| **2** | 3 | 0.02729 | 0.0 | 7.07 | 0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 242 | |
| **3** | 4 | 0.03237 | 0.0 | 2.18 | 0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 222 | |
| **4** | 5 | 0.06905 | 0.0 | 2.18 | 0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 222 | |

*d. Convert lstat from a percentage to a proportion. Name the new variable anything you wish. Verify by displaying the first six rows of the revised data frame.*

To do a variable transformation just enter the equation that defines the transformation. To convert a percentage to a proportion, divide by 100. Here name the new variable *lstat_prop*.

```
d['lstat_prop'] = d['lstat']/100
d.head()
```

| | Unnamed: 0 | crim | zn | indus | chas | nox | rm | age | dis | rad | tax | pt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.00632 | 18.0 | 2.31 | 0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1 | 296 | |
| **1** | 2 | 0.02731 | 0.0 | 7.07 | 0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2 | 242 | |
| **2** | 3 | 0.02729 | 0.0 | 7.07 | 0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2 | 242 | |
| **3** | 4 | 0.03237 | 0.0 | 2.18 | 0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3 | 222 | |
| **4** | 5 | 0.06905 | 0.0 | 2.18 | 0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3 | 222 | |

*e. Display just the average number of rooms for the second row of data.*

```
d.loc[1,'rm']
```

```
6.421
```

*f. To build a model to forecast median house price, analysts wish to focus on three predictor variables: crim, rm, and rad. Display the first five rows of data for just these three variables.*

*i. with the variable names*

```
d.loc[0:4, ['crim', 'rm', 'rad']]
```

| | crim | rm | rad |
|---|---|---|---|
| **0** | 0.00632 | 6.575 | 1 |
| **1** | 0.02731 | 6.421 | 2 |
| **2** | 0.02729 | 7.185 | 2 |
| **3** | 0.03237 | 6.998 | 3 |
| **4** | 0.06905 | 7.147 | 3 |

*ii. with the variable indices*

```
d.iloc[0:5, [1, 6, 9]]
```

| | crim | rm | rad |
|---|---|---|---|
| **0** | 0.00632 | 6.575 | 1 |
| **1** | 0.02731 | 6.421 | 2 |
| **2** | 0.02729 | 7.185 | 2 |
| **3** | 0.03237 | 6.998 | 3 |
| **4** | 0.06905 | 7.147 | 3 |

*g. List all the rows of data with the median value of the home less than 8000 USD.*

```
d.query('medv < 8')
```

| | Unnamed: 0 | crim | zn | indus | chas | nox | rm | age | dis | rad | tax |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **385** | 386 | 16.81180 | 0.0 | 18.10 | 0 | 0.700 | 5.277 | 98.1 | 1.4261 | 24 | 666 |
| **387** | 388 | 22.59710 | 0.0 | 18.10 | 0 | 0.700 | 5.000 | 89.5 | 1.5184 | 24 | 666 |
| **398** | 399 | 38.35180 | 0.0 | 18.10 | 0 | 0.693 | 5.453 | 100.0 | 1.4896 | 24 | 666 |
| **399** | 400 | 9.91655 | 0.0 | 18.10 | 0 | 0.693 | 5.852 | 77.8 | 1.5004 | 24 | 666 |
| **400** | 401 | 25.04610 | 0.0 | 18.10 | 0 | 0.693 | 5.987 | 100.0 | 1.5888 | 24 | 666 |
| **401** | 402 | 14.23620 | 0.0 | 18.10 | 0 | 0.693 | 6.343 | 100.0 | 1.5741 | 24 | 666 |
| **405** | 406 | 67.92080 | 0.0 | 18.10 | 0 | 0.693 | 5.683 | 100.0 | 1.4254 | 24 | 666 |
| **414** | 415 | 45.74610 | 0.0 | 18.10 | 0 | 0.693 | 4.519 | 100.0 | 1.6582 | 24 | 666 |
| **415** | 416 | 18.08460 | 0.0 | 18.10 | 0 | 0.679 | 6.434 | 100.0 | 1.8347 | 24 | 666 |
| **416** | 417 | 10.83420 | 0.0 | 18.10 | 0 | 0.679 | 6.782 | 90.8 | 1.8195 | 24 | 666 |
| **489** | 490 | 0.18337 | 0.0 | 27.74 | 0 | 0.609 | 5.414 | 98.3 | 1.7554 | 4 | 711 |

*h. Use code (i.e., do not manually count) to display the number of homes with median value < $8000.*

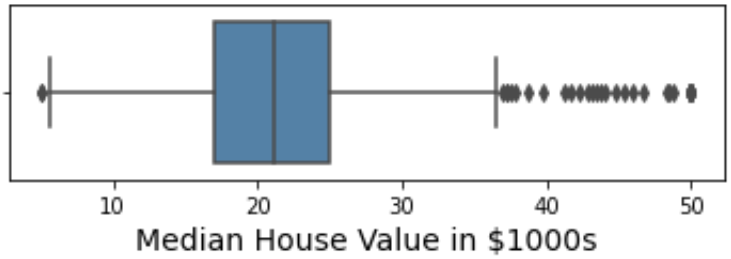
```
(d
 .query('medv <8')
 .shape[0]
)
```

```
      11
```

Note: `.shape` gives rows and columns of the data frame. The reference to `.shape[0]` gives just the first element, that is, the number of rows. Could also just do `.shape` and verbally indicate that the first element is the number of rows. Could also store in a separate data frame and then do `.shape` of that new data frame.

*i. Analysts want to build a model to forecast the median value of a house. Construct the box plot of the corresponding variable medv.*

```
plt.figure(figsize=(6,1.5))
sns.boxplot(x=d['medv'], color='steelblue')
plt.xlabel('Median House Value in $1000s', fontsize=14)
plt.show()
```



*j. Describe the distribution of medv from the box plot including any outliers.*

Half of the distribution consists of values between about 18,000 USD and 25,000 USD. Only a small number of values are larger than about 37,000 USD, which makes the distribution somewhat asymmetric. Moreover, those relatively small number of large values are outliers.

*k. For the three predictor variables of interest, rescale into a data object called X three ways, each time showing the first five rows of rescaled data.*

Going to need the `preprocessing` module from `sklearn`.

```
from sklearn import preprocessing
```

*i. MinMax, and also show the minimum and maximum of the rescaled variables.*

```
X = d[['crim', 'rm', 'rad']].copy()
X.head()
```

| | crim | rm | rad |
|---|---|---|---|
| **0** | 0.00632 | 6.575 | 1 |
| **1** | 0.02731 | 6.421 | 2 |
| **2** | 0.02729 | 7.185 | 2 |
| **3** | 0.03237 | 6.998 | 3 |
| **4** | 0.06905 | 7.147 | 3 |

```
mm_scaler = preprocessing.MinMaxScaler()
X = mm_scaler.fit_transform(X)
X = pd.DataFrame(X)
X.head()
```

| | 0 | 1 | 2 |
|---|---|---|---|
| **0** | 0.000000 | 0.577505 | 0.000000 |
| **1** | 0.000236 | 0.547998 | 0.043478 |
| **2** | 0.000236 | 0.694386 | 0.043478 |
| **3** | 0.000293 | 0.658555 | 0.086957 |
| **4** | 0.000705 | 0.687105 | 0.086957 |

```
X.min()
```

```
0    0.0
1    0.0
2    0.0
dtype: float64
```

```
X.max()
```

```
0    1.0
1    1.0
2    1.0
dtype: float64
```

All three variables re-scaled to range from 0 to 1.

*ii. Standardize, and also show the mean and standard deviation of the rescaled variables.*

```
X = d[['crim', 'rm', 'rad']].copy()
X.head()
```

| | crim | rm | rad |
|---|---|---|---|
| **0** | 0.00632 | 6.575 | 1 |
| **1** | 0.02731 | 6.421 | 2 |
| **2** | 0.02729 | 7.185 | 2 |
| **3** | 0.03237 | 6.998 | 3 |
| **4** | 0.06905 | 7.147 | 3 |

```
from sklearn.preprocessing import StandardScaler
s_scaler = preprocessing.StandardScaler()
```

```
Xst = s_scaler.fit_transform(X)
Xst = pd.DataFrame(Xst)
Xst.head()
```

|   | 0 | 1 | 2 |
|---|---|---|---|
| **0** | -0.419782 | 0.413672 | -0.982843 |
| **1** | -0.417339 | 0.194274 | -0.867883 |
| **2** | -0.417342 | 1.282714 | -0.867883 |
| **3** | -0.416750 | 1.016303 | -0.752922 |
| **4** | -0.412482 | 1.228577 | -0.752922 |

```
round(Xst.mean(), 4)
```

```
0   -0.0
1   -0.0
2    0.0
dtype: float64
```

The mean of a standardized variable is 0.

```
round(Xst.std(), 4)
```

```
0    1.001
1    1.001
2    1.001
dtype: float64
```

The standard deviation of a standardized variable is 1.

```
Xst.min()
```

```
0   -0.419782
1   -3.880249
2   -0.982843
dtype: float64
```

```
Xst.max()
```

```
0    9.933931
1    3.555044
2    1.661245
dtype: float64
```

There is a really large outlier for variable *crim* if anything close to a normal distribution. If normally distributed then most values should be within -3 and 3. It is not right or wrong for a distribution to be normal or not, but a z-score of almost 10 indicates either an extremem outlier or an strong asymetric distribution. ... These are the issues that need to be explored in data analysis before you begin the machine learning.

*iii. Robust Scale*

```
from sklearn.preprocessing import RobustScaler
r_scaler = preprocessing.RobustScaler()
```

```
Xrb = r_scaler.fit_transform(X)
Xrb = pd.DataFrame(Xrb)
Xrb.head()
```

| | 0 | 1 | 2 |
|---|---|---|---|
| **0** | -0.069593 | 0.496612 | -0.20 |
| **1** | -0.063755 | 0.287940 | -0.15 |
| **2** | -0.063760 | 1.323171 | -0.15 |
| **3** | -0.062347 | 1.069783 | -0.10 |
| **4** | -0.052144 | 1.271680 | -0.10 |

```
round(Xrb.mean(), 4)
```

```
0    0.9338
1    0.1032
2    0.2275
dtype: float64
```

```
round(Xrb.std(), 4)
```

```
0    2.3926
1    0.9521
2    0.4354
dtype: float64
```

```
round(Xrb.min(), 4)
```

```
0   -0.0696
1   -3.5874
2   -0.2000
dtype: float64
```

```
round(Xrb.max(), 4)
```

```
0    24.6784
1     3.4844
2     0.9500
dtype: float64
```

## Data Wrangling, Pre-Processing II

```
d = pd.read_excel("http://web.pdx.edu/~gerbing/data/SupermarketTransactions.x
```

The standard `d.head()` does not work well here because there are so many columns. Could transpose the display, but here just specified the first three rows and the ending columns. For the index of the ending column, just enter a large number to get all columns at the end.

```
d.iloc[0:3, 8:30]
```

| | City | State | Country | Family | Dept | Category | Units_Sold | Revenue |
|---|---|---|---|---|---|---|---|---|
| **0** | Los Angeles | CA | USA | Food | Snack Foods | Snack Foods | 5 | 27.38 |
| **1** | Los Angeles | CA | USA | Food | Produce | Vegetables | 5 | 14.90 |

*a. How many examples (rows of data) are there in the data file?*

```
d.shape[0]
```

```
14059
```

*b. Convert the value of Country, USA, to USofA.*

```
d = d.replace({'Country': {'USA': 'USofA'}})
d.iloc[0:3, 8:30]
```

| | City | State | Country | Family | Dept | Category | Units_Sold | Revenue |
|---|---|---|---|---|---|---|---|---|
| **0** | Los Angeles | CA | USofA | Food | Snack Foods | Snack Foods | 5 | 27.38 |
| **1** | Los Angeles | CA | USofA | Food | Produce | Vegetables | 5 | 14.90 |

*c. Sales took place in three countries. Convert the categorical variable Country to dummy variables for later numerical processing.*

One of the dummy variables needs to be dropped before ready for later analysis.

```
d = pd.get_dummies(d, columns=['Country'], drop_first=True)
d.iloc[0:3, 8:20]
```

| | City | State | Family | Dept | Category | Units_Sold | Revenue | Country |
|---|---|---|---|---|---|---|---|---|
| **0** | Los Angeles | CA | Food | Snack Foods | Snack Foods | 5 | 27.38 | |
| **1** | Los Angeles | CA | Food | Produce | Vegetables | 5 | 14.90 | |

## Missing Data

```
d = pd.read_excel("http://web.pdx.edu/~gerbing/data/employee.xlsx")
```

*a. How many samples (rows of data) are there in the data file?*

```
d.shape[0]
```

```
37
```

*b. Display rows of data that include the row of data with the missing data.*

```
d[d.isna().any(axis='columns')]
```

| | Name | Years | Gender | Dept | Salary | JobSat | Plan | Pre | Post |
|---|---|---|---|---|---|---|---|---|---|
| **1** | Wu, James | NaN | M | SALE | 84494.58 | low | 1 | 62 | 74 |
| **3** | Jones, Alissa | 5.0 | F | NaN | 43772.58 | NaN | 1 | 65 | 62 |
| **30** | Korhalkar, Jessica | 2.0 | F | ACCT | 62502.50 | NaN | 2 | 74 | 87 |

*c. Impute the median for the missing data of Years employed at the company.*

Will do the transformation on a subset of the *d* data frame, here named X, which is consistent with machine learning, where X contains the predictor variables, the features for the subsequent machine learning.

A programming "gotcha", however, is that subsetting from a data frame of only a single variable, here *Years*, results in a `pandas` data structure called a `Series`, a single column in a `pandas` data frame, instead of a data frame of only one column. Convert this `Series` back to a full data frame with the `pandas` function `DataFrame()`. Then display the missing data for James Wu for variable *Years*.

When things go wrong, check the types of your data structures. There are several more that we do not cover in this class. The type of structure is a fundamental data concept.

```
X = d.loc[:, 'Years']
X = pd.DataFrame(X)
X.iloc[1, 0]
```

```
nan
```

```
from sklearn.impute import SimpleImputer
imp_med = SimpleImputer(missing_values=np.nan, strategy='median')
imp_med = imp_med.fit(X)
X = imp_med.transform(X)
```

*d. Display rows of data that include the row of data with the imputed data to verify that the missing data has been properly imputed to show the change from missing to the imputed median for each variable.*

```
X = pd.DataFrame(X)
X.iloc[1,0]
```

```
9.0
```