National Research University Higher School of Economics

Faculty of Computer Science

Master of Science in Data Science

Titanic: Machine Learning from Disaster

Project of Modern Data Analysis

Writing by: Fouzi Takelait & Anton Broilovskiy

Problem Definition:

The sink of *RMS*[1] *Titanic* is one of the most infamous shipwrecks in history. On April 12, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

The reason these data make an interesting project that we want to attempt and discover during this study.

Research question:

1. What sorts of people were likely to survive the tragedy?
2. Are 3rd class passengers more unlikely to survive than those of 1st and 2nd class?
3. What kind of passengers are more similar to be survived?

Data Description:

Table 1 displays row 1, 2, 3 and 183 of a datasets (Kaggle, 2012) concerning 183 passengers were in the ship (hence the name of the dataset). These observations will be referred to as the Titanic data set, and they are a random sample from a larger dataset[2].

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38	1	0	PC 17599	71.2833	C85	C
2	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
3	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
…	…	…	…	…	…	…	…	…	…	…	…
183	1	1	Behr, Mr. Karl Howell	male	26	0	0	111369	30	C148	C

Table 1 Four rows from RMS Titanic data matrix.

[1] **Royal Mail Ship**, is the ship prefix used for seagoing vessels that carry mail under contract to the British Royal Mail.

[2] These data were originally collected by the British Board of Trade in their investigation of the sinking. (Dawson, 1995)

These data set have the following dimensions: 183 ***cases***[3] and 12 variables. Descriptions of all ten passengers' variables are given in Table 2.

Variable	Definition	Key
Survival	Survival	0 = No, 1 = Yes
Pclass	Ticket class purchased by the passenger	1 = 1st , 2 = 2nd, 3 = 3rd
Sex	Sex	Male, Female
Age	Age in years	
SibSp	# of siblings / spouses aboard the Titanic	
Parch	# of parents / children aboard the Titanic	
Ticket	Ticket number	
Fare	Passenger fare	
Cabin	Cabin number	
Embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Table 2 Variables, descriptions and their keys for the Titanic data set.

Comments on some variable

[3] A case is also sometimes called a **unit of observation** or an **observational unit**.

Pclass: A proxy for socio-economic status (SES)

- 1st = Upper
- 2nd = Middle
- 3rd = Lower

Age: Age is fractional if less than 1. If the age is estimated, it is in the form of xx.5

SibSp: The data set defines family relations in the following way:

- Sibling = brother, sister, stepbrother, stepsister,
- Spouse = husband, wife (mistresses and fiancés were ignored).

Parch: The data set defines family relations in the following way:

- Parent = mother, father,
- Child = daughter, son, stepdaughter, stepson,
- Some children travelled only with a nanny, therefore parch = 0 for them

Each row in the table represents a single passenger or ***case****.* The columns represent characteristics, called ***variables***, for each of the passengers. For example, the first row represents the passenger Mrs. John Bradley, a 38 years old woman, being a 1st class passenger and has been embarked from Cherbourg's port who survived.

Homework 2, I

K-Means clustering:

1. **Choose 3-6 features, Explain the choice, Apply K-means:**
   - **At K=5**
   - **At K=9**
   - **In both cases: 10 or more random initializations, chose the best over the K-means criterion**

In this part, we're going to implement the hard clustering method called K-Means using the Titanic dataset. Before proceeding with it, we might be thinking that since it is labelled dataset, how could it be used for a clustering task?

We just have to drop **survival** column, which is the dependent variable, from the dataset and make it unlabeled. It's the task of K-Means to cluster the records of the datasets if they survived or not.

For this specific task, we are going to work only with 4 variables: **Age**, **Fare**, **SibSp** and **Parch**.

Implementation:

Throughout this report, we'll use the Python programming language for implementing all our machine learning algorithms given in homework. Our choice of this language is based in several reasons which we site some of them as follows: (Christopher Brooks, Kevyn Collins-Thompson, Daniel Romero, & V. G. Vinod Vydiswaran, s.d.)

1. It's easy to learn
   - Now the language of choice for 8 of 10 top US computer science programs (Philip Guo, CACM)
2. Full featured
   - Not just a statistics language, but has full capabilities for data acquisition, cleaning databases, high-performance computing, and more
3. Strong Data Science Libraries
   - The SciPy Ecosystem

For making our job done we need the following Python packages: Pandas, NumPy, Scikit-Learn, Seaborn and Matplotlib. (See: Appendix)

```
# importing necessary packages
import pandas as pd
import numpy as np
import matplotlib as plt
%matplotlib inline

# reading Titanic dataset from hard drive
df = pd.read_csv('train.csv')
```

It is very important to note that not all machine learning algorithms support *missing values* in the data that we are feeding to them. K-Means being one of them. So we need to handle missing values present in the data which ours already have not.

Data exploration:

First, we need to do some analytics in order to understand the data better. Understanding our data is really required in order to perform any machine learning task. Let's start finding out which features are categorical and which are numerical (See Table 3).

Categorical		Quantitative	
Nominal	Ordinal	Continuous	Discrete
Survived	Pclass	Age	SibSp
Sex		Fare	Parch
Embarked			

Table 3 Titanic variables defined in different categories.

Two features are left out which are not listed above in any of the categories. Which are **Ticket** and **Cabin**. **Ticket** is a mix of *numeric* and *alphanumeric* data types. **Cabin** is *alphanumeric*. Let see some sample values.

```
# showing the first 5 data points of the Ticket variable.
df.Ticket.head()

```

```
# Output:
0    PC 17599
1      113803
2       17463
3     PP 9549
4      113783
Name: Ticket, dtype: object
```

```
# showing the first 5 data points of the Cabin variable.
df.Cabin.head()

# Output:
0     C85
1    C123
2     E46
3      G6
4    C103
Name: Cabin, dtype: object
```

We observe that features like **Name**, **Ticket**, **Cabin** and **Embarked** do not have any impact on the survival status of the passengers. It is better to train our model with only significant features than to train it with all the features, including unnecessary ones. It not only helps in efficient modelling, but also the training of the model can happen in much lesser time.

Again, the features we have selected to train our M-Means model are **Age**, **SibSp**, **Parch**, and **Fare**. The reason for this choice is that these features are considered as they have a huge impact on the **Survived** status of the Titanic passengers. These are quantitative variables the reason is in case of selecting binary features thill will create a maximum variance will let other features (non-binary ones) follows this feature.

Before training our K-Means model we need to do some data transformation for our remaining variables. Before converting them into numeric one, we need to do some *feature engineering*. Since we only have the feature **Sex** which is a categorical variable (non-numeric) then we can apply the Label Encoding[4] technique for this latter.

```python
# importing Label Encoder module from scikit-learn library
from sklearn.preprocessing import LabelEncoder

# dropping all unecessery features from our data frame
df.drop(['Name', 'Ticket', 'Cabin', 'Embarked'], axis=1, inplace=True)

# setting an instance (object) LabelEncoder()
le = LabelEncoder()
# Fitting the object with categorical variable
sex_le = le.fit(df['Sex'])
# Converting 'Sex' feature to numerical one
df['Sex'] = le.transform(df['Sex'])

# print the first 5 cases of our data (see Table 4)
df.head()
```

Age	Fare	SibSp	Parch
38	71.283	1	0
35	53.1	1	0
54	51.863	0	0
4	16.7	1	1
58	26.55	0	0

Table 4 Data Matrix ready to be trained with K-Means algorithm

[4] Encode labels with value between 0 and n_classes-1 (i.e. **Sex** feature has two classes whether *male* or *female* then these two will be encoding 1 for male and 0 for female).

Again, in this case, we will cluster the feature **Age**, **Fare**, **SibSp**, and **Parch** into 5 and 9 clusters. First, we need to check if these features do not contain any missing values using the following command.

```
data = df.copy()
data = data.loc[:, [‘Age’, ‘Fare’, ‘SibSp’, ‘Parch’]]
np.isnull(data).sum()
# Output
Age    0
Fare   0
SibSp 0
Parch 0
dtype: int64
```

2. Interpret each found partition by using features from the data table. Explain why you consider one of them better than the other in this perspective.

```
rom sklearn.cluster import KMeans
def kmeans(df):
    SSE = []
    for cluster in [5, 9]:
        kmeans = KMeans(n_clusters=cluster, init='random', n_init=10)
        kmeans.fit(df)
        pred = kmeans.predict(df)
        SSE.append(kmeans.score(df))
    return {'score of 5 clusters': round(SSE[0],3), 'score of 9 clusters': round(SSE[1],3)}

kmeans(data)
# Output
{'score of 5 clusters': -187578.824, 'score of 9 clusters': -40331.864}
```

This results means that the less clusters we have, for this data, the better score we git.

Homework 2, II

Task: Take a feature, find the 95% confidence interval for its grand mean by using bootstrap.

Confidence interval represents the frequency of possible confidence intervals that contain the true value of the unknown population parameter. In other words, if confidence intervals are constructed using a given confidence level from an infinite number of independent sample statistics, the proportion of those intervals that contain the true value of the parameter will be equal to the confidence level. In our case, the confidence level z is equal 95% or in other words critical value z = $-\Phi^{-1}(1 - \alpha/2)$, where Φ is CDF of Standard normal distribution.

The confidence interval is calculated by this formula: $(\boldsymbol{E} - \boldsymbol{z} * \frac{\boldsymbol{Var}}{\sqrt{\boldsymbol{N}}}; \boldsymbol{E} + \boldsymbol{z} * \frac{\boldsymbol{Var}}{\sqrt{\boldsymbol{N}}})$, where E - Math expectation, Var - variance, N - the number of elements in the dataset.

In this task, the dataset for confidence interval will be found by the bootstrap algorithm or bootstrapping. The main idea is that one can model the data by resampling the data from the dataset and performing statistic parameters about a dataset such as global mean or variance using only resampled data.

We will find the 95% confidence interval for feature **Age**. And firstly we need to take a samples from the dataset with replacement. Each sample will have size N equal the dataset size. In total M = 5000 samples are generated.

```
age = df.Age
N = age.shape[0]
M = 5000
bootstap_age = []
# data sampling
for i in range(M):
    indices = np.random.choice(range(N), size=N)
    bootstap_age.append(age[indices])
bootstap_age = np.concatenate(bootstap_age)
```

The Below you can see the calculating of global mean and variance on sampled data, and finding the 95% confidence interval.

```
bs_mean = bootstap_age.mean()
```

```
bs_std  = bootstap_age.std()
z = 1.96 # critical value for 95% confidence interval
print("95% confidence interval for feature 'Age' is: ({:.3}; {:.3})".format(bs_mean + z*bs_std/N**.5, bs_mean - z*bs_std/N**.5))

Output:
95% confidence interval for feature 'Age' is: (37.9; 33.4)
```

Homework 3

Contingency Table:

1. **Consider three nominal features (one of them, not more, may be taken from nominal features in your data)**
2. **Build two contingency tables over them: present a conditional frequency table and Quetelet relative index tables. Make comments on relations between categories of the common (to both tables) feature and two others.**

Discovering relationships between variables is the fundamental goal of data analysis. Frequency tables are a basic tool we can use to explore data and get an idea of the relationships between variables. A frequency table is just a data table that shows the counts of one or more categorical variables. In this study we will use the two-way frequency tables, also called contingency tables, and are tables of counts with two dimensions where each dimension is a different variable.

We will select for this task three variables which are: **Sex**, **Pclass** and **Survived** (***Note***: We have to select only nominal features), where we will draw two conditional frequency tables, the first will draw the relationship (resp. Quetelet index) between the variables **Sex** and **Survived** whereas the second one will draw the relationship (resp. Quetelet index) between the variables **Survived** and **Pclass**.

The following Python code will output the conditional frequency table for the variables **Sex** and **Survived**.

```
# Table of survival vs sex
survived_Sex = pd.crosstab(index=df.Survived, columns=df.Sex, margins=True)

survived_Sex.index = ['died', 'survived', 'coltotal']

# conditional frequency table for the variables Sex and Survived
```

```
conditional_Sex  = survived_Sex / survived_Sex.loc['coltotal']
```

The following Python code will output the conditional frequency table for the variables **Servived** and **Pclass** (passenger class).

```
# Table of survival vs passenger class
survived_class = pd.crosstab(index=df["Survived"], columns=df["Pclass"], margins=True)

survived_class.columns = ["class1","class2","class3", 'rowtotal']
survived_class.index= ["died","survived", 'coltotal']

# conditional frequency table for the variables survived and Pclass
conditional_class = survived_class / survived_class.loc['coltotal']
```

Sex	female	male
died	0.06818	**0.56842**
survived	**0.93182**	0.43158

Table 5. Conditional frequency table of features Sex and Survived

Pclass	class 1	class 2	class 3
died	0.32911	0.2	**0.5**
survived	**0.67089**	**0.8**	**0.5**

Table 6. Conditional frequency table of features Survived and Pclas

From the left-hand side table, we can conclude that females are more likely to be survived that males whereas in the right-hand side table we can conclude that the proportion of survival in the passenger classes is higher than to those who died.

3. **Compute and visualize the chi-square-summary_Quetelet_index over both tables. Comment on the meaning of the values in the data analysis context.**

Quetelet indices for the same previous features are as follows respectively:

```
# quetelet index for sex variable
proportion_survived = survived_Sex / 183
quetelet_indice_sex = conditional_Sex / proportion_survived - 1
quetelet_indice_sex
```

```
# quetelet index for passenger class variable
proportion_survived = survived_class / 183
quetelet_indice_class = conditional_class / proportion_survived - 1
quetelet_indice_class
```

Sex	female	male
died	1.07955	0.92632
survived	1.07955	0.92632

Table 7 Quetelet index for Sex variable

	class1	class2	class3
died	0.15823	11.2	17.3
survived	0.15823	11.2	17.3

Table 8 Quetelet index for Pclass variable

Homework 4

PCA/SVD

4.1. In your data set, select a subset of 3-6 features related to the same aspect and explain your choice.

In this task, we will apply the PCA algorithm to our dataset and check the influence of PCA on the target feature interpretation with two different types of normalization. Therefore, we must choose only numerical features. In the dataset, we have two numerical features - **Age** and **Fare**. And to fulfill the conditions specified in the task, we must to create a new numerical feature using the previous one. We had created a feature named **Age_Fare** with the following value: $'Age—Fare' = 50 * |\, sin('Age'.value) + cos('Fare')|$.

```
df['Age_Fare'] = 50*np.abs(np.sin(df['Age']) + np.cos(df['Fare']))
std_df = df[['Age', 'Fare', 'Age_Fare']].copy()
```

4.2. Standardize the selected subset; compute its data scatter and determine contributions of all the principal components to the data scatter, naturally and per cent.

Standardization is a process of putting different variables on the same scale. This process allows you to compare scores between different types of variables. This process consist two parts:

- shift the origin to m.
- rescaling data by retailing to b.

And the main formula is the following: $Y = \frac{(X - a)}{b}$, where usually **a** is mean value of **X**, **b** is range of **X** or standard deviation of **X**.

In the propose to compute all variants these functions were written:

```
std_wh_stdiv = lambda df: (df - df.mean())/(df.std())
std_wh_range = lambda df: (df - df.mean())/(df.shape[0])
```

Let’s standardizate all features in our dataset using standard deviation normalization.

```
std_df['Age'] = std_wh_stdiv(std_df['Age'])
std_df['Fare'] = std_wh_stdiv(std_df['Fare'])
std_df['Age_Fare'] = std_wh_stdiv(std_df['Age_Fare'])
```

Data scatter calculated by sum of the square values of the dataset: $DS = \sum_{i,j} (X^2)$.

```
std_val = std_df.values
scatter = np.sum(std_val*std_val)
print("Data scatter: {:.3}".format(scatter))

Output:
Data scatter: 5.46e+02
```

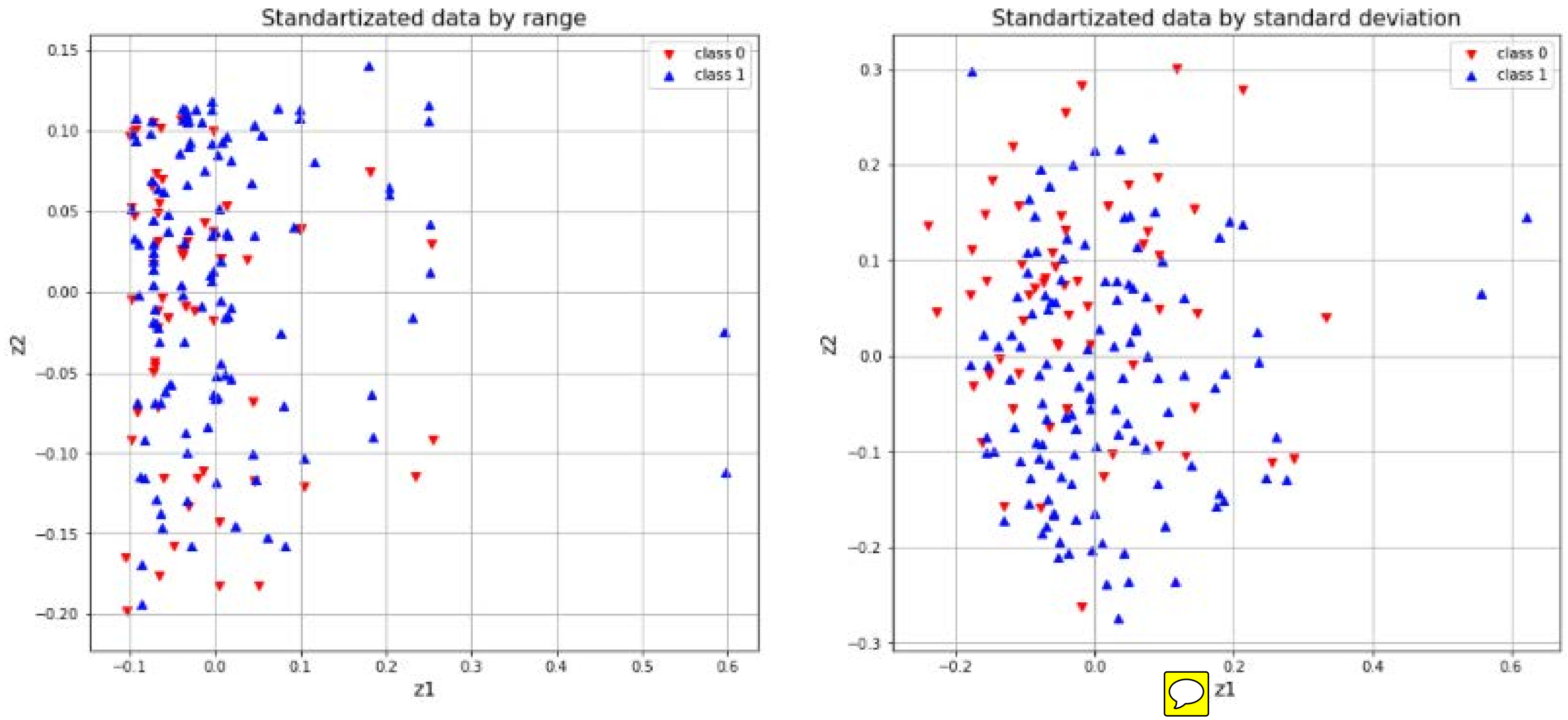
Contributions of all the principal components could be calculated using SVD and data scatter: $PC = \frac{mu^2}{DS}$. To calculate a per cent value of PC you need to multiply value by 100. PC_per = PC * 100.

```
mu = np.linalg.svd(std_val)[1]
num = ['first', 'second', 'third']
for u, s in zip(mu, num):
    comp = u**2/scatter
    print('Contribution of the {} principal component:\t{:.3} = {:.3}%'.format(s, comp, comp*100))

Output:
Contribution of the first principal component:  0.385 = 38.5%
Contribution of the second principal component: 0.336 = 33.6%
Contribution of the third principal component:  0.279 = 27.9%
```

4.3. Visualize the data with these features using standardization with two versions of normalization: (a) over ranges and (b) over standard deviations. At these visualizations, use a distinct shape/colour for points representing a pre-specified by you group of objects. Also, apply the conventional PCA for the visualization and see if there is any difference. Comment on which of the normalizations is better and why.

From the lections we know that the best 2D approximation of the data it is use only two first singular triplets. Let's visualize two variants of the data:



a) Standardization data with normalizations over ranges. b) Standardization data with normalizations over standard deviations. The color and shape of the data depend on the **Survived** feature. You can see below the code of visualization.

```
_, ax = plt.subplots(1, 2, figsize=(18, 8))
sr = ['range', 'standard deviation']

for q, std in enumerate([std_rg_df, std_sd_df]):
    x1 = std.values
    y1 = np.array([x1[:,i] - x1[:,i].mean() for i in range(len(x1.T))]).T
    z, mu, c = np.linalg.svd(y1)
    z1 = z[:,0] * (mu[0])**.2
    z2 = z[:,1] * (mu[1])**.2

    ax[q].set_title('Standartizated data by %s'% sr[q], fontdict={'fontsize':15})
    first_z = df.index[df['Survived']==0].tolist()
    second_z = df.index[df['Survived']==1].tolist()
    ax[q].scatter(z1[first_z], z2[first_z], c='r', marker='v', label='class 0')
    ax[q].scatter(z1[second_z], z2[second_z], c='b', marker='^', label='class 1')
    ax[q].set_xlabel('z1', fontdict={'fontsize':14})
    ax[q].set_ylabel('z2', fontdict={'fontsize':14})
    ax[q].legend()
    ax[q].grid()
```

PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables. Conventional PCA another approach that consist three parts:

1. Compute covariance matrix by centroid version of main matrix.

   $B = \frac{Y^TY}{N}$, where B - covariance matrix, Y - centroid matrix, N - size of Y.

```
x = std_sd_df.values
y1 = np.array([x[:,i] - x[:,i].mean() for i in range(len(x.T))]).T
B = (y1.T.dot(y1))/len(y1)
```

2. Compute eigenvalues and eigenvectors of B.

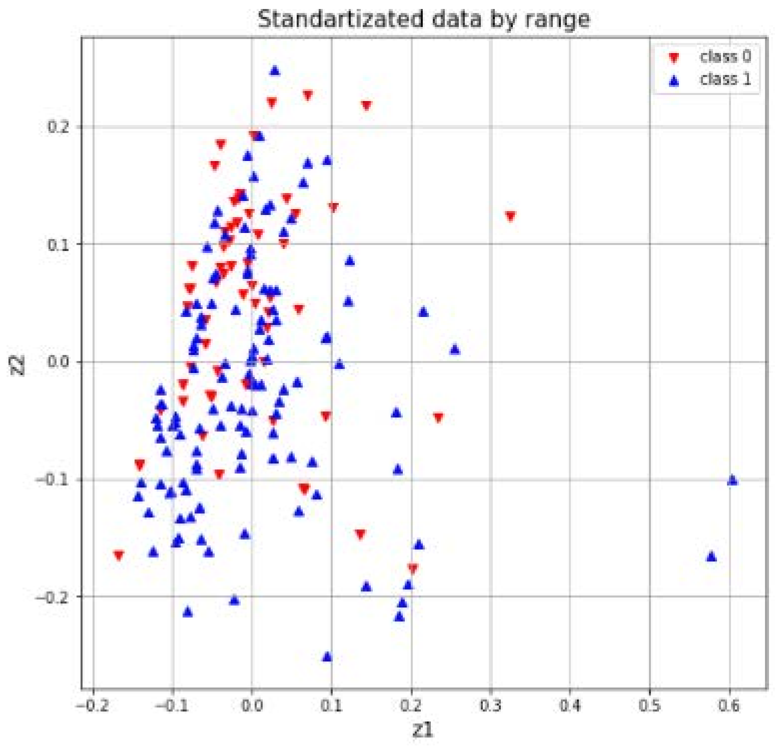
```
la, c = np.linalg.eigh(B)
```

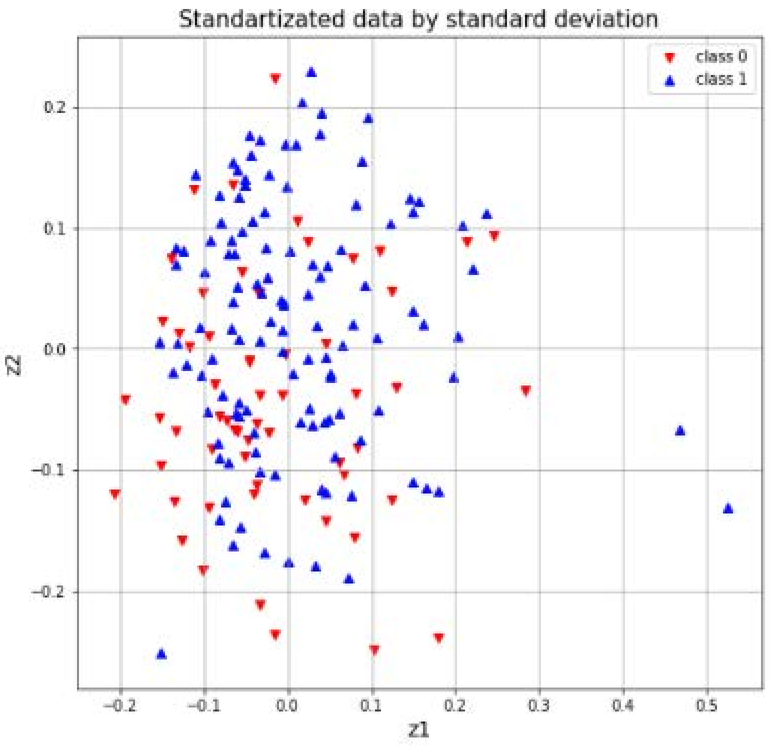
3. Compute the principal component scoring matrix.

   Each vector in matrix equal $zi = \frac{Yci}{\sqrt{6*\lambda i}}$.

```
Z = []
for l, com in zip(la[::-1], c[::-1]):
   Z.append((y1.dot(com))/(6*l)**.5)
```

Now let’s draw once more and discuss the results.





As you can see, the data with standard deviation normalization have bigger distance between elements of different classes. It is means that this way of normalization have better influence in data, than normalization by range.

4.4. Compute and interpret a hidden factor behind the selected features. The factor should be expressed in a 0-100 rank scale (as well as the features).

The hidden factor allows you to inference your data features and find the most useful feature. To find it you need to make 3 actions:

1. Normalize features so that maximum gets a value of 100.

```
sb = df[['Age', 'Fare', 'Age_Fare']].copy().values
m = np.max(sb)
n_sb = sb * 100 / m
```

2. Find first singular triplet with positive loadings.

```
z, mu, c = np.linalg.svd(n_sb)
c1 = -c[:,0]
```

3. Determine the hidden factor.

```
a = 1/np.sum(c1)
print('Hidden factor: {:.3}'.format(a))

Output:
Hidden factor: 0.632
```

And now lets calculate Z using PCA and find the evaluate the contribution of each feature.

```
c = np.abs(np.linalg.svd(n_sb)[-1][:,0])
print("Z = {:.3}*'Age' + {:.3}* 'Fare' + {:.3}*'Age_Fare'".format(*c*a))

Output:
Z = 0.15*'Age' + 0.334* 'Fare' + 0.515*'Age_Fare'
```

As you can see, the most useful component is ‘Age_Fare’. It happening because this feature contains the rest features.

Homework 5

Linear regression:

- **1. Find two features in your dataset with more or less “linear-like” scatterplot.**
- **2. Display the scatter-plot.**
- **3. Build a linear regression of one of the features over the other. Make a comment on the meaning of the slope.**

Scientific question:

What is the estimated value for the variable *Fare* when a passenger’s *Age* is 85 and/or 18?

In this part of our study, we will show the relationship between two numerical (continuous) variables. For this, a correlation plot will show us how these two variables are correlated. Besides this will draw a trend line (linear regression) to help make the relationship more clear.

We will select the following two variables from our dataset: **Fare** and **Age**, and try to build a linear model as follows:

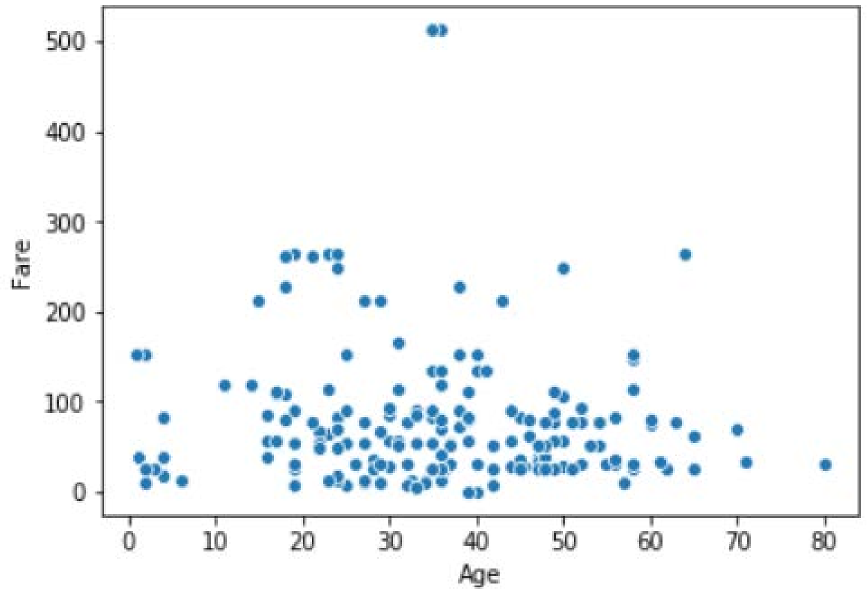
Fare = slope*Age + intercept

Such that **Fare** is the response variable and **Age** is our predictor.

First, we will plot a mathematical diagram using Cartesian coordinates to display values for typically two selected variables as follows:

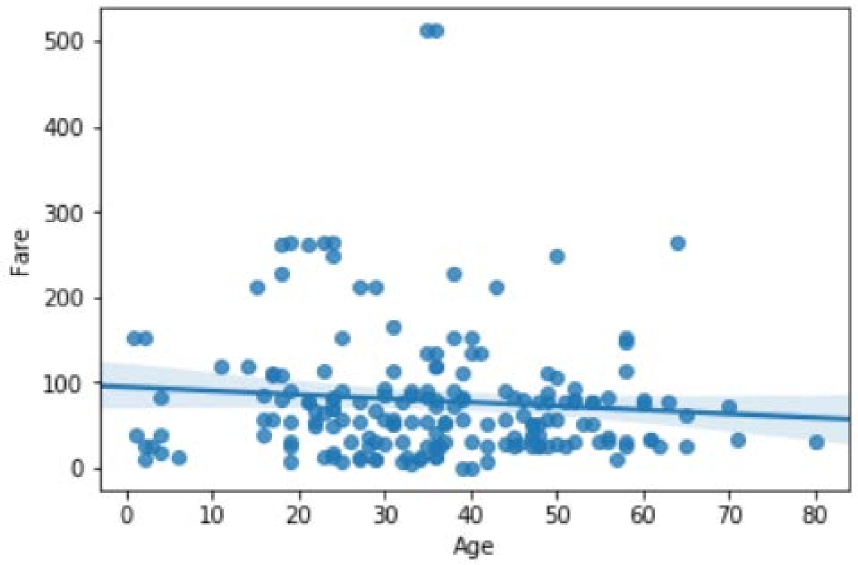
```
# importing seaborn Library
import seaborn as sns

# plotting the scatter plot such that Age variable will be in x-axis and Fare variable in y-axis
scatterplot = sns.scatterplot(x='Age', y='Fare', data=df);
```



Now we can draw our linear regression using the following Python code:

```python
# Fitting the linear model on the data
reg = sns.regplot(x='Age', y='Fare', data=df)
```



4. **Find the correlation and determinacy coefficients, and comment on the meaning of the latter.**
5. **Make a prediction of the target values for given two or three predictor' values; make a comment**
6. **Compare the mean relative absolute error of the regression on all points of your set and the determinacy coefficient and make comments**

As we can visually, our slope seems to be less than zero which leads that our correlation coefficient is negative which we will determine it latter. By now, we need to determine the parameters *slope* and *intercept* of our linear model as follows:

```python
# importing scientific python library
from scipy import stats

# calculating the slope, the intercept and the correlation coefficient for our model
slope, intercept, r_value, p_value, std_error = stats.linregress(df.Age, df.Fare)


# slope, intercept
```

```
round(slope, 3), round(intercept, 3)
# Output
(-0.451, 94.774)
```

Our model will be as follow:

Fare = -0.451*Age + 94.774

Where the correlation coefficient is equal to: **R = -9.242**

```
# calculating R
round(df.Age.corr(df.Fare),3)*100

# Output
-9.242
```

Interpretation of R: In statistics, the correlation coefficient *R* measures the strength and direction of a linear relationship between two variables on a **scatterplot**. The value of *R* is always between +1 and –1. We have **R = -9.242 ~ -0.30** Then we have weak downhill (negative) linear relationship as pretended. This means that the variables *Fare* and *Age* are negatively correlated (i.e. if *Age* increase then *Fare* decrease).

After doing all this job, we want to answer our research question asked on the beginning of this part which is: "What is the estimated value for the variable *Fare* when a passenger's *Age* is 85 and/or 18?"

Answering this question now becomes easy as we already have the linear model with estimated parameters (*Slope* and *intercept*) we need just to assign 85 and 18 to the variable *Age*.

For Age = 85 we have Fare = -0.451*85 + 94.774 evaluating this gives us **Fare = 56.434**.

Same for Age = 18 we got **Fare = 86.655**.

These results show exactly our expectations about the relationship between these two variables which is: "If *Age* increase then *Fare* decrease" which is definitely true.

Mean square error: The mean square error is a risk metric corresponding to the expected value of the absolute error loss. If *Age* is the predicted value of the i^{th} sample, and *Fare* is the corresponding true value, then the mean absolute value (MAE) estimated over $N_{samples}$ is defined as:

$$MAE(Fare, Age) = \frac{1}{Nsamples} \sum_{i=0}^{Nsamples-1} |Fare(i) - Age(i)|$$

In fact, we use this metric to measure the forecast accuracy of our model.

```
# importing the module metrices from the Library sklearn
from sklearn.metrics import mean_absolute_error

# Fare vector data
y_true = [x for x in df.Fare]

# Age vector data
y_pred = [x for x in slope*df.Age+intercept]

# calculating the MAE
mean_absolute_error(y_true, y_pred)

# Output
50.084
```

Our Mean Absolute Value is equal to **50.084**. This value can be interpreted as the forecast accuracy of our linear model is 50% of our sample dataset. **Note:** a forecast has been accurate in the past does not mean it will be accurate in be future.

Bibliography

Christopher Brooks, Kevyn Collins-Thompson, Daniel Romero, & V. G. Vinod Vydiswaran. (n.d.). *Applied Data Science with Python Specialization*. Retrieved from Coursera.org: https://www.coursera.org/specializations/data-science-python

Dawson, R. J. (1995). The “Unusual Episode” Data Revisited. *Journal of Statistics Education, 3*. doi:10.1080/10691898.1995.11910499

Kaggle. (2012, September 28). *Getting Started Prediction Competition*. Retrieved from https://www.kaggle.com/c/titanic.

Appendix

Pandas

pandas is a Python data analysis library which is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Sckit-learn

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.