Machine Learning A-Z

Q & A

All the answers to your questions, section by section

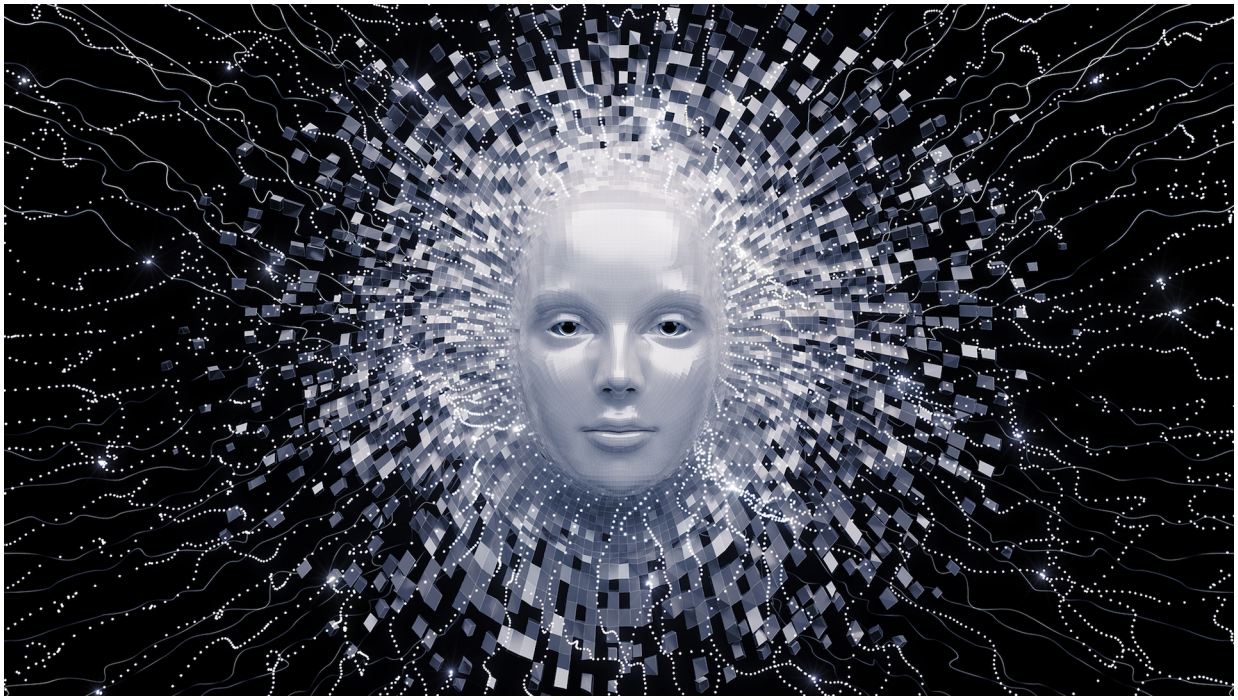


Table of contents

1 Part 1 - Data Preprocessing

1.1 Importing the dataset

I can't import the dataset. It says that the file is not found. What should I do?
Python: Make sure that in File Explorer, you are in the folder that contains the file 'Data.csv' which you want to import. This folder is called the "Working Directory folder".
R: Make sure that you set the right working directory folder as we do in the Lecture and that this working directory folder contains the file 'Data.csv'.

What is the difference between the independent variables and the dependent variable?
The independent variables are the input data that you have, with each you want to predict something. That something is the dependent variable.

In Python, why do we create X and y separately?
Because we want to work with Numpy arrays, instead of Pandas dataframes. Numpy arrays are the most convenient format to work with when doing data preprocessing and building Machine Learning models. So we create two separate arrays, one that contains our independent variables (also called the input features), and another one that contains our dependent variable (what we want to predict).

In Python, what does 'iloc' exactly do?
It locates the column by its index. In other words, using 'iloc' allows us to take columns by just taking their index.

In Python, what does '.values' exactly do?
It returns the values of the columns you are taking (by their index) inside a Numpy array. That is basically how X and y become Numpy arrays.

In R, why don't we have to create arrays?
Because R works very differently than Python. R contains great tools that allow to work directly and easily with dataframes.

1.2 Missing Data

In Python, what is the difference between fit and transform?
The fit part is used to extract some info of the data on which the object is applied (here, Imputer will spot the missing values and get the mean of the column). Then, the transform part is used to apply some transformation (here, Imputer will replace the missing value by the mean).

In R, why do we use the ave() function when replacing the missing values by a mean of the column when we can do it much simpler this way:

```
dataset$Age[is.na(dataset$Age)] = mean(dataset$Age, na.rm = T)
```

We use the ave() function with the FUN parameter because it allows us to add some options in the calculation of the mean, like for example calculating the mean of observations grouped by another feature, or calculating the mean of subsets, etc. This function can be very useful if you want to be more accurate when replacing the missing values by the mean.

Is replacing by the mean the best strategy to handle missing values?
It is a good strategy but not the best one always. It depends on your business problem, on the way your data is distributed and on the number of missing values. If for example you have a lot of missing values, then mean substitution is not the best thing. Other strategies include "median" imputation, "most frequent" imputation or prediction imputation. Prediction Imputation is actually another great strategy that is recommended but that I didn't cover in Part 1 because it was too advanced to do it in Part 1. This strategy indeed requires to understand Part 3 - Classification. So if you completed Part 3, here is the strategy that is even better than mean imputation: you take your feature column that contains the missing values and you set this feature column as the dependent variable, while setting the other columns as the independent variables. Then you split your dataset into a Training set and a Test set where the Training set contains all the observations (the lines) where your feature column that you just set as the dependent variable has no missing value and the Test set contains all the observations where your dependent variable column contains the missing values. Then you perform a classification model (a good one for this situation is k-NN) to predict the missing values in the test set. And eventually you replace your missing values by the predictions. A great strategy!

1.3 Categorical Data

In Python, what do the two 'fit_transform' methods do?
When the 'fit_transform()' method is called from the LabelEncoder() class, it transforms the categories strings into integers. For example, it transforms France, Spain and Germany into 0, 1 and 2. Then, when the 'fit_transform()' method is called from the OneHotEncoder() class, it creates separate columns for each different labels with binary values 0 and 1. Those separate columns are the dummy variables.

In R, why don't we manually create the dummy variables like we do in Python
Because they are automatically created when using the 'factor()' function as we do in the Lecture. We will visually see that when starting regression and classification.

1.4 Splitting the dataset into the Training set and Test set

What is the difference between the training set and the test set?
The training set is a subset of your data on which your model will learn how to predict the dependent variable with the independent variables. The test set is the complimentary subset from the training set, on which you will evaluate your model to see if it manages to predict correctly the dependent variable with the independent variables.

Why do we split on the dependent variable?
Because we want to have well distributed values of the dependent variable in the training and test set. For example if we only had the same value of the dependent variable in the training set, our model wouldn't be able to learn any correlation between the independent and dependent variables.

1.5 Feature Scaling

Do we really have to apply Feature Scaling on the dummy variables?
Yes, if you want to optimize the accuracy of your model predictions.
No, if you want to keep the most interpretation as possible in your model.

When should we use Standardization and Normalization?
Generally you should normalize (normalization) when the data is normally distributed, and scale (standardization) when the data is not normally distributed. In doubt, you should go for standardization. However what is commonly done is that the two scaling methods are tested.

2 Part 2 - Regression

2.1 Simple Linear Regression

2.1.1 Simple Linear Regression Intuition

What are exactly the coefficients b_0 and b_1 in the Simple Linear Regression equation: 'Salary $= b_0 + b_1 \times$ Experience'
b_0 is the salary you get with no experience and b_1 is the increase in salary per year.

Why do we take the squared differences and simply not the absolute differences?
Because the squared differences makes it easier to derive a regression line. Indeed, to find that line we need to compute the first derivative of the loss error function, and it is much harder to compute the derivative of absolute values than squared values.

2.1.2 Simple Linear Regression in Python

Why didn't we apply Feature Scaling in our Simple Linear Regression model?
It's simply because since y is a linear combination of the independent variables, the coefficients can adapt their scale to put everything on the same scale. For example if you have two independent variables x_1 and x_2 and if y takes values between 0 and 1, x_1 takes values between 1 and 10 and x_2 takes values between 10 and 100, then b_1 can be multiplied by 0.1 and b_2 can be multiplied by 0.01 so that y, $b_1 x_1$ and $b_2 x_2$ are all on the same scale.

What does 'regressor.fit(X_train, y_train)' do exactly?
The fit method will take the values of X_train and y_train and then will compute the coefficients b_0 and b_1 of the Simple Linear Regression equation ($y = b_0 + b_1 x$) as seen in the Intuition Lecture. That's the whole purpose of this fit method here.

2.1.3 Simple Linear Regression in R

What is the p-value?
To understand the P-value, we need to start by understanding the null hypothesis: the null hypothesis is the assumption that the parameters associated to your independent variables are equal to zero. Therefore under this hypothesis, your observations are totally random, and don't follow a certain pattern. The P-value is the probability that the parameters associated to your independent variables have certain nonzero values, given that the null hypothesis is True. The most important thing to keep in mind about the P-Value is that it is a statistical metric: the lower is the P-Value, the more statistically significant is an independent variable, that is the better predictor it will be.

In the last R lecture we see a gray area around the blue line. What does it mean and how can we get it?
The gray area around the fit line is the confidence interval. You can add '+ geom_smooth(method = 'lm')' right after 'geom_line(...)' to obtain these confidence boundaries.

How to get the P-Values in Python like we do in R?
You will learn how to do that in the next section on Multiple Linear Regression.

2.2 Multiple Linear Regression

2.2.1 Multiple Linear Regression Intuition

What are the Multiple Linear Regression assumptions in more details?
Linearity: There must be a linear relationship between the dependent variable and the independent variables. Scatterplots can show whether there is a linear or curvilinear relationship.
Homoscedasticity: This assumption states that the variance of error terms is similar across the values of the independent variables. A plot of standardized residuals versus predicted values can show whether points are equally distributed across all values of the independent variables.
Multivariate Normality: Multiple Linear Regression assumes that the residuals (the differences between the observed value of the dependent variable y and the predicted value $\hat{y}$ are normally distributed.
Independence of errors: Multiple Linear Regression assumes that the residuals (the differences between the observed value of the dependent variable y and the predicted value $\hat{y}$ are independent.
Lack of multicollinearity: Multiple Linear Regression assumes that the independent variables are not highly correlated with each other. This assumption is tested using Variance Inflation Factor (VIF) values.

How is the coefficient b_0 related to the dummy variable trap?
Since $D_2 = 1 - D_1$ then if you include both D_1 and D_2 you get:

$$\begin{aligned} y &= b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4 D_1 + b_5 D_2 \\ &= b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4 D_1 + b_5 (1 - D_1) \\ &= b_0 + b_5 + b_1 x_1 + b_2 x_2 + b_3 x_3 + (b_4 - b_5) D_1 \\ &= b_0^* + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4^* D_1 \end{aligned}$$

with $b_0^* = b_0 + b_5$ and $b_4^* = b_4 - b_5$
Therefore the information of the redundant dummy variable D_2 is going into the constant b_0.

2.2.2 Multiple Linear Regression in Python

We predicted the outcomes of a set of observations (the test set). How do we do the same for a single observation?
Let's say this observation has the following features: Feature Value 1, Feature Value 2, ... and Feature Value m (m independent variables). Then the code to get the predicted outcome would be the following:

```
y_pred = regressor.predict(np.array([[Feature Value 1, ..., Feature Value m]]))
```

How to implement automatic Backward Elimination in Python?
You can implement it this way:

```
import statsmodels.formula.api as sm
def backwardElimination(x, sl):
    numVars = len(x[0])
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    x = np.delete(x, j, 1)
    regressor_OLS.summary()
    return x
```

```
SL = 0.05
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
X_Modeled = backwardElimination(X_opt, SL)
```

2.2.3 Multiple Linear Regression in R

We predicted the outcomes of a set of observations (the test set). How do we do the same for a single observation?
You first need to create a new 'single_observation' variable with the input values, and setting this variable as a dataframe:

```
single_observation = data.frame(R.D.Spend = 10000,
                                Administration = 20000,
                                Marketing.Spend = 30000,
                                State = 1)
```

Then you can simply predict the outcome of this single observation exactly as we did with the test set, by just replacing 'test_set' by 'single_observation':

```
y_pred = predict(regressor, newdata = single_observation)
```

How to implement automatic Backward Elimination in R?
You can implement it this way:

```
backwardElimination <- function(x, sl) {
    numVars = length(x)
    for (i in c(1:numVars) ){
      regressor = lm(formula = Profit ~ ., data = x )
      maxVar = max(coef(summary(regressor))[c(2:numVars), "Pr(>|t|)"])
      if (maxVar > sl){
        j = which(coef(summary(regressor))[c(2:numVars),"Pr(>|t|)"]==maxVar)
        x = x[,-j]
      }
      numVars = numVars - 1
    }
    return(summary(regressor))
  }

  SL = 0.05
  dataset = dataset[,c(1,2,3,4,5)]
  backwardElimination(dataset, SL)
```

2.3 Polynomial Regression

2.3.1 Polynomial Regression Intuition

Is Polynomial Regression a linear or non linear model?
That depends on what you are referring to. Polynomial Regression is linear on the coefficients since we don't have any power of the coefficients (all the coefficients are raised to the power of 1: b_0, b_1, ..., b_n). However, Polynomial Regression is a non linear function of the input x, since we have the inputs raised to several powers: x (power 1), x^2 (power 2), ..., x^n (power n). That is how we can also see the Polynomial Regression as a non linear model. Besides indeed, Polynomial Regression is appropriate when the data is non linearly distributed (meaning you can't fit a straight line between y and x).

2.3.2 Polynomial Regression in Python

Why didn't we apply Feature Scaling in our Polynomial Regression model?
It's simply because, since y is a linear combination of x and x^2, the coefficients can adapt their scale to put everything on the same scale. For example if y takes values between 0 and 1, x takes values between 1 and 10 and x^2 takes values between 1 and 100, then b_1 can be multiplied by 0.1 and b_2 can be multiplied by 0.01 so that y, b_1x_1 and b_2x_2 are all on the same scale.

How do we find the best degree?
The main form of finding a good fit is to plot the model and see what it looks like visually. You simply test several degrees and you see which one gives you the best fit. The other option is to find the lowest root-mean-square error (RMSE) for your model, but in that case be careful not to overfit the data.

Why did we have to create a second linear regressor 'lin_reg_2'? Could we not have used 'lin_reg' directly?
No, because 'lin_reg' is already fitted to X and y and now we want to fit a new linear model to X_poly and y. So we have to create a new regressor object. One must important that the fit method here finds the coefficient between the independent variables and the dependent variable. Therefore since 'lin_reg' already got the coefficients of correlation between X and y, 'lin_reg_2' has to be created to get some new coefficients of correlations between X_poly and y.

2.3.3 Polynomial Regression in R

In R, I have to create manually the different columns for each polynomial feature? What if there are many polynomial features?
You will very rarely have to create more than 4 polynomial features, otherwise you would get overfitting, which must absolutely be avoided in Machine Learning. So even if you have to create them manually, it will never take too much of your time. Besides you can use the template.

How do we find the best degree? (Asking this question again in case some students only do R
The main form of finding a good fit is to plot the model and see what it looks like visually. You simply test several degrees and you see which one gives you the best fit. The other option is to find the lowest root-mean-square error (RMSE) for your model, but in that case be careful not to overfit the data.

2.4 SVR

2.4.1 SVR Intuition

When should I use SVR?
You should use SVR if a linear model like linear regression doesn't fit very well your data. This would mean you are dealing with a non linear problem, where your data is not linearly distributed. Therefore in that case SVR could be a much better solution.

I didn't understand the Intuition Lecture. Am I in trouble?
Not at all. SVR is a pretty abstract model and besides it is not that commonly used. What you must rather understand is the SVM model, which you will see in Part 3 - Classification. Then once you understand the SVM model, you will get a better grasp of the SVR model, since the SVR is simply the SVM for Regression. However we wanted to include SVR in this course to give you an extra option in your Machine Learning toolkit.

2.4.2 SVR in Python

Why do we need to 'sc_Y.inverse_transform'?
We need the inverse_transform method to go back to the original scale. Indeed we applied feature scaling so we get this scale around 0 and if we make a prediction without inversing the scale we will get the scaled predicted salary. And of course we want the real salary, not the scaled one, so we have to use 'sc_Y.inverse_transform'. Also what is important to understand is that 'transform' and 'inverse_transform' are paired methods.

2.4.3 SVR in R

Why did we not apply feature scaling like we did explicitly in Python
That's because in svm() function of R, the values are automatically scaled.

Can we select the most significant variables thanks to the p-value like we did in R before?
You couldn't use p-value because SVR is not a linear model, and p-values apply only to linear models. Therefore feature selection is out of the question. But you could do feature extraction, which you will see in Part 9 - Dimensionality Reduction. That you can apply to Decision Trees, and it will reduce the number of your features.

2.5 Decision Tree Regression

2.5.1 Decision Tree Regression Intuition

How does the algorithm split the data points?
It uses reduction of standard deviation of the predictions. In other words, the standard deviation is decreased right after a split. Hence, building a decision tree is all about finding the attribute that returns the highest standard deviation reduction (i.e., the most homogeneous branches).

What is the Information Gain and how does it work in Decision Trees?
The Information Gain in Decision Tree Regression is exactly the Standard Deviation Reduction we are looking to reach. We calculate by how much the Standard Deviation decreases after each split. Because the more the Standard Deviation is decreased after a split, the more homogeneous the child nodes will be.

What is the Entropy and how does it work in Decision Trees?
The Entropy measures the disorder in a set, here in a part resulting from a split. So the more homogeneous is your data in a part, the lower will be the entropy. The more you have splits, the more you have chance to find parts in which your data is homogeneous, and therefore the lower will be the entropy (close to 0) in these parts. However you might still find some nodes where the data is not homogeneous, and therefore the entropy would not be that small.

2.5.2 Decision Tree Regression in Python

Does a Decision Tree make much sense in 1D?
Not really, as we saw in the practical part of this section. In 1D (meaning one independent variable), the Decision Tree clearly tends to overfit the data. The Decision Tree would be much more relevant in higher dimension, but keep in mind that the implementation we made here in 1D would be exactly the same in higher dimension. Therefore you might want to keep that model in your toolkit in case you are dealing with a higher dimensional space. This will actually be the case in Part 3 - Classification, where we will use Decision Tree for Classification in 2D, which you will see turns out to be more relevant.

2.5.3 Decision Tree Regression in R

Why do we get different results between Python and R?
The difference is likely due to the random split of data. If we did a cross-validation (see Part 10) on all the models in both languages, then you would likely get a similar mean accuracy. That being said, we would recommend more using Python for Decision Trees since the model is slightly better implemented in Python.

Is the Decision Tree appropriate here?
Here in this example, we can clearly see that the fitting curve is a stair with large gaps in the discontinuities. That decision tree regression model is therefore not the most appropriate, and that is because we have only one independent variables taking discrete values. So what happened is that the prediction was made in the lower part of the gap in Python, and made in the upper part of the gap in R. And since the gap is large, that makes a big difference. If we had much more observations, taking values with more continuity (like with a 0.1 step), the gaps would be smaller and therefore the predictions in Python and R far from each other.

Can we select the most significant variables thanks to the p-value like we did in R before?
You couldn't use p-value because Decision Tree is not a linear model, and p-values apply only to linear models. Therefore feature selection is out of the question. But you could do feature extraction, which you will see in Part 9 - Dimensionality Reduction. That you can apply to Decision Trees, and it will reduce the number of your features.

2.6 Random Forest Regression

2.6.1 Random Forest Regression Intuition

What is the advantage and drawback of Random Forests compared to Decision Trees? Advantage: Random Forests can give you a better predictive power than Decision Trees.
Drawback: Decision Tree will give you more interpretability than Random Forests, because you can plot the graph of a Decision Tree to see the different splits leading to the prediction, as seen in the Intuition Lecture. That's something you can't do with Random Forests.

When to use Random Forest and when to use the other models?
The best answer to that question is: try them all!
Indeed, thanks to the templates it will only take you 10 minutes to try all the models, which is very little compared to the time dedicated to the other parts of a data science project (like Data Preprocessing for example). So just don't be scared to try all the regression models and compare the results (through cross validation which we will see in Part 10). That's we gave you the maximum models in this course for you to have in your toolkit and increase your chance of getting better results.
However then, if you want some shortcuts, here are some rules of thumbs to help you decide which model to use:
First, you need to figure out whether your problem is linear or non linear. You will learn how to do that in Part 10 - Model Selection. Then: If your problem is linear, you should go for Simple Linear Regression if you only have one feature, and Multiple Linear Regression if you have several features.
If your problem is non linear, you should go for Polynomial Regression, SVR, Decision Tree or Random Forest. Then which one should you choose among these four? That you will learn in Part 10 - Model Selection. The method consists of using a very relevant technique that evaluates your models performance, called k-Fold Cross Validation, and then picking the model that shows the best results. Feel free to jump directly to Part 10 if you already want to learn how to do that.

2.6.2 Random Forest Regression in Python

How do I know how many trees I should use?
First, I would recommend to choose the number of trees by experimenting. It usually takes less time than we think to figure out a best value by tweaking and tuning your model manually. That's actually what we do in general when we build a Machine Learning model: we do it in several shots, by experimenting several values of hyperparameters like the number of trees. However, also know that in Part 10 we will cover k-Fold Cross Validation and Grid Search, which are powerful techniques that you can use to find the optimal value of a hyperparameter, like here the number of trees.

2.6.3 Random Forest Regression in R

How do we decide how many trees would be enough to get a relatively accurate result?
Same as in Python, you can find a relevant number of tree through experimentation and manual tweaking. You should use enough trees to get a good accuracy, but you shouldn't use too many trees because that could cause overfitting. You will learn how to find the optimal number of trees in the first section of Part 10 - Model Selection. It's done with a technique called Parameter Tuning (Grid Search with k-Fold Cross Validation).

Why do we get different results between Python and R?
The difference is likely due to the random split of data. If we did a cross-validation (see Part 10) on all the models in both languages, then you would likely get a similar mean accuracy.

Can we select the most significant variables thanks to the p-value like we did in R before?
You couldn't use p-value because Random Forests are not linear models, and p-values apply only to linear models. Therefore feature selection is out of the question. But you could do feature extraction, which you will see in Part 9 - Dimensionality Reduction. That you can apply to Random Forests, and it will reduce the number of your features.

2.7 Evaluating Regression Models Performance

I understand that in order to evaluate multiple linear models I can use the Adjusted R-squared or the Pearson matrix. But how can I evaluate Polynomial Regression models or Random Forest Regression models which are not linear?
You can evaluate polynomial regression models and random forest regression models, by computing the "Mean of Squared Residuals" (the mean of the squared errors). You can compute that easily with a sum function or using a for loop, by computing the sum of the squared differences between the predicted outcomes and the real outcomes, over all the observations of the test set.
You couldn't really apply Backward Elimination to Polynomial Regression and Random Forest Regression models because these models don't have coefficients combined in a linear regression equation and therefore don't have p-values.
However in Part 9 - Dimensionality Reduction, we will see some techniques like Backward Elimination, to reduce the number of features, based on Feature Selection & Feature Extraction. The course is still in development and right now we are working on Part 8 - Deep Learning, but Part 9 - Dimensionality Reduction will come out quickly afterwards.

What are Low/High Bias/Variance in Machine Learning?
Low Bias is when your model predictions are very close to the real values.
High Bias is when your model predictions are far from the real values.
Low Variance: when you run your model several times, the different predictions of your observation points

won't vary much.
High Variance: when you run your model several times, the different predictions of your observation points will vary a lot.
What you want to get when you build a model is: Low Bias and Low Variance.

In Python, how to implement automated Backward Elimination with Adjusted R-Squared?
Let's take again the problem of the Multiple Linear Regression, with 5 independent variables. Automated Backward Elimination including Adjusted R Squared can be implemented this way:

```
import statsmodels.formula.api as sm
def backwardElimination(x, SL):
    numVars = len(x[0])
    temp = np.zeros((50,6)).astype(int)
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        adjR_before = regressor_OLS.rsquared_adj.astype(float)
        if maxVar > SL:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    temp[:,j] = x[:, j]
                    x = np.delete(x, j, 1)
                    tmp_regressor = sm.OLS(y, x).fit()
                    adjR_after = tmp_regressor.rsquared_adj.astype(float)
                    if (adjR_before >= adjR_after):
                        x_rollback = np.hstack((x, temp[:,[0,j]]))
                        x_rollback = np.delete(x_rollback, j, 1)
                        print (regressor_OLS.summary())
                        return x_rollback
                    else:
                        continue
    regressor_OLS.summary()
    return x

SL = 0.05
X_opt = X[:, [0, 1, 2, 3, 4, 5]]
X_Modeled = backwardElimination(X_opt, SL)
```

How to get the Adjusted R-Squared in R?
Let's denote the dependent variable by DV, and the independent variables by IV1, IV2, etc. Then the R code to get the Adjusted R-Squared is the following:

```
summary(lm(DV ~ IV1 + IV2 + ..., dataset))$adj.r.squared
```

3 Part 3 - Classification

3.1 Logistic Regression

3.1.1 Logistic Regression Intuition

Is Logistic Regression a linear or non linear model?
It is a linear model. You will visualize this at the end of the section when seeing that the classifier's separator is a straight line.

What are the Logistic Regression assumptions?
First, binary logistic regression requires the dependent variable to be binary and ordinal logistic regression requires the dependent variable to be ordinal.
Second, logistic regression requires the observations to be independent of each other. In other words, the observations should not come from repeated measurements or matched data.
Third, logistic regression requires there to be little or no multicollinearity among the independent variables. This means that the independent variables should not be too highly correlated with each other.
Fourth, logistic regression assumes linearity of independent variables and log odds. although this analysis does not require the dependent and independent variables to be related linearly, it requires that the independent variables are linearly related to the log odds.

Can Logistic Regression be used for many independent variables as well?
Yes, Logistic Regression can be used for as many independent variables as you want. However be aware that you won't be able to visualize the results in more than 3 dimensions.

3.1.2 Logistic Regression in Python

What does the fit method do here?
The fit method will basically train the Logistic Regression model on the training data. Therefore it will compute and get the weights (coefficients) of the Logistic Regression model (see the Intuition Lecture) for that particular set of training data composed of X_train and y_train. Then right after it collects the weights/coefficients, you have a Logistic Regression model fully trained on your training data, and ready to predict new outcomes thanks to the predict method.

We predicted the outcomes of a set of observations (the test set). How do we do the same for a single observation, to predict a single outcome?
Let's say this observation has the following features: Age = 30, Estimated Salary = 50000.
Then the code to get the predicted outcome would be the following (notice how we must not forget to scale that single observation first):

```
y_pred = classifier.predict(sc_X.transform(np.array([[20, 50000]])))
```

Is the Confusion Matrix the optimal way to evaluate the performance of the model?
No, it just gives you an idea of how well your model can perform. If you get a good confusion matrix with few prediction errors on the test set, then there is a chance your model has a good predictive power. However the most relevant way to evaluate your model is through K-Fold Cross Validation, which you will see in Part 10. It consists of evaluating your model on several test sets (called the validation sets), so that we can make sure we don't get lucky on one single test set. Today most Data Scientists or AI Developers evaluate their model through K-Fold Cross Validation. However the technique is a different subject so I preferred to leave it for Part 10 after we cover all the different models.

How to re-transform Age and Salary back to their original scale?
You can re-transform Age and Salary by using the inverse_transform() method. You take you scaler object "sc" and you apply the inverse_transform() method on it this way:

```
X_train = sc.inverse_transform(X_train)
X_test = sc.inverse_transform(X_test)
```

How to do the same classification when the dependent variables has more than two classes?
Let's explain how to do it with three classes.
First, it is still the same code to make the predictions. Besides although it is based on the One vs All method, you don't even need to create dummy variables of your dependent variable.
As for the independent variables, you just need to apply LabelEncoder followed by OneHotEncoder. It will create your dummy variables, whatever the number of categories your categorical variables have.
And eventually, you simply need to add a colour this way:

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1,
                               step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                               stop = X_set[:, 1].max() + 1,
                               step = 0.01))
plt.contourf(X1,
             X2,
             classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75,
             cmap = ListedColormap(('red', 'green', 'blue')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0],
                X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

As you can see in this code, I just added the 'blue' colour, which I had to do because since we now have three classes, then np.unique(y_set) becomes 3.

Can you explain the meshgrid() method in more details and some of its uses?
In the meshgrid() method, you input two arguments:
First argument is the range values of the x-coordinates in your grid.
Second is the range values of the y-coordinates in your grid.
So let's say that these 1st and 2nd arguments are respectively [-1,+1] and [0,10], then you will get a grid where the values will go from [-1,+1] on the x-axis and [0,10] on the y-axis.

Can you explain the contourf() method in more details and some of its uses?
Before using the contourf method, you need to build a grid. That's what we do in the line just above when building X1 and X2.
Then, the contourf() method takes several arguments:
1. The range values of the x-coordinates of your grid,
2. The range values of the y-coordinates of your grid,
3. A fitting line (or curve) that will be plotted in this grid (we plot this fitting line using the predict function because this line are the continuous predictions of our model),
4. Then the rest are optional arguments like the colours to plot regions of different colours.
The regions will be separated by this fitting line, that is in fact the contour line.

3.1.3 Logistic Regression in R

We predicted the outcomes of a set of observations (the test set). How do we do the same for a single observation?
You first need to create a new 'single_observation' variable with the input values (for example Age = 30 and Estimated Salary = 50000), and setting this variable as a dataframe:

```
single_observation = data.frame(Age = 30, EstimatedSalary = 50000)
```

Then you can simply predict the outcome of this single observation exactly as we did with the test set, by just replacing 'test_set' by 'single_observation':

```
prob_pred = predict(classifier, type = 'response', newdata = single_observation)
y_pred = ifelse(prob_pred > 0.5, 1, 0)
```

How to inverse the scaling in R?
You can do it with the following code:

```
training_set[-3] * attr(training_set[-3], 'scaled:scale')
+ attr(training_set[-3], 'scaled:center')
test_set[-3] * attr(test_set[-3], 'scaled:scale')
+ attr(training_set[-3], 'scaled:center')
```

How to visualize the same classification when the dependent variables has more than two classes?
Let's suppose we have for example three categories (or classes). Then the code for visualization would be the following:

```
# Visualizing the Training set results
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('PC1', 'PC2')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = 'SVM (Training set)',
     xlab = 'PC1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
```

```
points(grid_set,
       pch = '.',
       col = ifelse(y_grid == 2,'deepskyblue', ifelse(y_grid == 1, 'springgreen3', 'tomato')))
points(set,
       pch = 21,
       bg = ifelse(set[, 3] == 2,
       'blue3', ifelse(set[, 3] == 1,
       'green4',
       'red3')))
```

3.2 K-Nearest Neighbors (K-NN)

3.2.1 K-NN Intuition

Is K-NN a linear model?
No, K-NN is a non linear model, as you will see in the practical sections of this course.

What if we found equal number of neighbors? For example, if we choose K=6 and if we find 3 data points in each category, where should we fit the new data point?
The point is then assigned randomly to one of the two.

What kind of business problems requires K-NN?
K-NN can be a good solution for Natural Language Processing. You will see that in Part 7. Also, K-NN works well for non linear problems (when the data is non linearly separable), as you will see in this section (Python or R).

3.2.2 K-NN in Python

What is the meaning of n = 5 when building the K-NN model?
n = 5 means that you will train the K-NN model with 5 neighbours. When you watch the Intuition tutorial for K-NN, n = 5 is the number of neighbors Kirill mentions in the videos.

What number of neighbors should we choose?
The more you have neighbors, the more this team of neighbors has chance to find correct predictions, and therefore the more your model accuracy has chance to increase. However be careful, if you have too many neighbors, that will cause overfitting on the training set and the predictions will be poor on new observations in the test set.

How can I inverse the scaling to get the original scale of the features?
You can re-transform Age and Salary by using the inverse_transform() method. You take you scaler object "sc" and you apply the inverse_transform() method on it this way:

```
X_train = sc.inverse_transform(X_train)
X_test = sc.inverse_transform(X_test)
```

3.2.3 K-NN in R

How to visualize the output in the form of graph if the number of predictors is more than 2?
You could not because one predictor corresponds to one dimension. So if we have more than 2 predictors, it would be hard to represent the output visually. If we have three predictors, we could still do it using some

advanced packages and in that case the prediction regions would become some prediction volumes (you know region spaces in 3 dimensions), and the prediction boundary would no longer be a fitting curve, but would become some curved plan separating the prediction volumes. It would be a little harder to see that visually but still possible. But then in 4+ dimensions, that would be very hard.

What number of neighbors should we choose?
The more you have neighbors, the more this team of neighbors has chance to find correct predictions, and therefore the more your model accuracy has chance to increase. However be careful, if you have too many neighbors, that will cause overfitting on the training set and the predictions will be poor on new observations in the test set.

3.3 Support Vector Machine (SVM)

3.3.1 SVM Intuition

Is SVM a linear model?
Yes, SVM is a linear model. You will see that easily in the practical sections of this course, when visualizing the results on the graph (you will notice that the prediction boundary is a straight line). However we can make the SVM a non linear model, by adding a kernel, which you will see in the next section.

Why does we see the support vectors as vectors not as points?
The vectors are points in 2-D space (as in this example), but in real-world problems we have data-sets of higher dimensions. In an n-dimensional space, vectors make more sense and it is easier to do vector arithmetic and matrix manipulations rather than considering them as points. This is why we generalize the data-points to vectors. This also enables us to think of them in an N-dimensional space.

3.3.2 SVM in Python

What does the fit method do here?
It will simply train the SVM model on X_train and y_train. More precisely, the fit method will collect the data in X_train and y_train, and from that it will compute the support vectors. Once the support vectors are computed, your classifier model is fully ready to make new predictions with the predict method (because it only requires the support vectors to classify new data).

3.3.3 SVM in R

If my dataset has more than 2 independent variables, how do I plot the visualization?
In that case you could not plot the visualization, because one independent variable corresponds to one dimension. So if you had for example 8 independent variables, you would need to create a plot in 8 dimensions. That would be impossible. However you can use dimensionality reduction techniques to reduce the number of independent variables, by extracting the most statistically significant ones. So if you manage to reduce your 8 original variables down to two, you can then plot the visualization in 2D.

Why do I get different results in Python and R?
The difference can come from a variety of factors:
- the random factors in the model and the fact we use different seeds,
- the different default values of the hyperparameters (the parameters that are not learned) but you say they are the same,
- slight differences in the algorithms across Python and R.

3.4 Kernel SVM

3.4.1 Kernel SVM Intuition

Why exactly are we converting the high dimensional space in 3D back to 2D?
That's because we need to go back to our original space that contains our independent variables. If you stayed in the 3d space you would loose the information of your independent variables because in this 3d space there are not your original independent variables but the projections of them. So you want to go back to the original 2d space by projecting back the points.

When we apply the transformation $f = x - 5$ the points just move left on the same 1D axis. But when we apply the transformation $f = (x - 5)^2$, why do the points move to a U curve in 2D? Should the points not stay on the same axis in 1D?
Two different kinds of transformation are involved here, the first one is a transformation in one dimension for the coordinate x, and should be seen this way:

$$x' = x - 5$$

With this transformation (called translation), the points are moved 5 units to the left. Then, the mapping transformation occurs, which is the mapping transformation involved in Kernel SVM. But this one should be seen this way:

$$y = (x' - 5)^2$$

where y is the new coordinate you create with this mapping in higher dimension.

Which Kernel to choose?
A good way to decide which kernel is the most appropriate is to make several models with different kernels, then evaluate each of their performance, and finally compare the results. Then you choose the kernel with the best results. Be careful to evaluate the model performance on new observations (preferably with K-Fold Cross Validation that we will see in Part 10) and to consider different metrics (Accuracy, F1 Score, etc.).

3.4.2 Kernel SVM in Python

What does the fit method do here?
It will simply train the SVM model on X_train and y_train with a non linear Kernel. More precisely, the fit method will collect the data in X_train and y_train, and from that it will do all the successive operations you saw in the Intuition Lecture: first, a mapping in a higher dimensional space where the data is linearly separable, then compute the support vectors in this higher dimensional space, and eventually a projection back into 2D.

How to predict the outcome of a single new observation?
You need to input the data of your single observation in an array, scale it with our sc scaler object (we need to scale here because our model was trained on scaled data), and use the predict method like this:

```
single_prediction = classifier.predict(sc.transform(np.array([[30, 80000]])))
```

3.4.3 Kernel SVM in R

How to know easily with R if the dataset is not linearly separable?
If it is a regression problem, make a multiple linear regression model with all your independent variables included, and if it is a classification problem, make a logistic regression model with all your independent

variables included. Then use the stats summary() function in R to see the statistical significance levels of your independent variables. If some of them have statistical significance (small p-values), then your dataset is likely to be linearly separable. If not, then your dataset might not be linearly separable.

We predicted the outcomes of a set of observations (the test set). How do we do the same for a single observation?
You first need to create a new 'single_observation' variable with the input values (for example Age = 30 and Estimated Salary = 50000), and setting this variable as a dataframe:

```
single_observation = data.frame(Age = 30, EstimatedSalary = 50000)
```

Then you can simply predict the outcome of this single observation exactly as we did with the test set, by just replacing 'test_set' by 'single_observation':

```
y_pred = predict(classifier, newdata = single_observation)
```

How to inverse the scaling in R?
You can do it with the following code:

```
training_set[-3] * attr(training_set[-3], 'scaled:scale')
+ attr(training_set[-3], 'scaled:center')
test_set[-3] * attr(test_set[-3], 'scaled:scale')
+ attr(training_set[-3], 'scaled:center')
```

3.5 Naive Bayes

3.5.1 Naive Bayes Intuition

Is Naive Bayes a linear model or a non linear model?
Naive Bayes is a non linear model. You will see that very clearly in Python or R when plotting the prediction boundary which will be a very nice curve well separating the non linearly distributed observations.

Can $P(x)$ be zero?
Yes $P(X)$ can be zero. That can happen when the new data point is an outlier, and therefore there is no other data points within the radius of the circle. However the formula in the Naive Bayes theorem is only true when $P(X)$ is different than zero. When $P(X)$ is equal to zero, then $P(Walks|X)$ is just set to zero.

How does the algorithm decide the circle?
In the Intuition lecture we see that a circle is drawn to create a collection of data points similar to the new datapoint. The new datapoint was roughly at the center of the circle and hence we saw that number of green points were lesser than the number of red points and hence the new point went to the red category. But if we had drawn the circle a little differently around the new datapoint, then the number of green points could have been more than red. So how is that circle chosen? There is a parameter in the model that decides the radius of the circle, just like there is a parameter that chooses the number of neighbors in K-NN.

Naive Bayes vs K-NN
The best way to answer that question is to try them both. Thanks to the templates it will only take a few minutes to compare their results. Just understand that the main difference between the two is that Naive Bayes is based on a probabilistic approach, unlike classic K-NN.

3.5.2 Naive Bayes in Python

What does the fit method do here?
It will simply train the Naive Bayes model on X_train and y_train. More precisely, the fit method will collect the data in X_train and y_train, and from that it will do all the successive operations you saw in the Intuition Lecture: first it will get the prior probability, the marginal likelihood, the likelihood and the posterior probability, and then using the circles around the observations it will learn how to classify these observations and predict future outcomes.

How to get the probabilities as output?
Here is the way to do that:

```
y_proba = classifier.predict_proba(X_test)
```

This will give you an array with 2 columns, one for 0 and one for 1.

3.5.3 Naive Bayes in R

Why do we need to convert the dependent variable as a factor?
We do this to specify that the dependent variable is a categorical variable and that 0 & 1 should be treated as categories (factors).

In the naiveBayes() function, is using the argument 'formula = Purchased ~ .' similar to specifying x and y as we did in the video?
Yes that would be the same but you would need to specify the data argument. Basically you must specify what is the dependent variable, the independent variable and the training set. That's sufficient information for the function to know how to train the model.

3.6 Decision Tree Classification

3.6.1 Decision Tree Classification Intuition

How does the algorithm split the data points?
It uses reduction of standard deviation of the predictions. In other words, the standard deviation is decreased right after a split. Hence, building a decision tree is all about finding the attribute that returns the highest standard deviation reduction (i.e., the most homogeneous branches).

What is the Information Gain and how does it work in Decision Trees?
The Information Gain in Decision Tree Regression is exactly the Standard Deviation Reduction we are looking to reach. We calculate by how much the Standard Deviation decreases after each split. Because the more the Standard Deviation is decreased after a split, the more homogeneous the child nodes will be.

What is the Entropy and how does it work in Decision Trees?
The Entropy measures the disorder in a set, here in a part resulting from a split. So the more homogeneous is your data in a part, the lower will be the entropy. The more you have splits, the more you have chance to find parts in which your data is homogeneous, and therefore the lower will be the entropy (close to 0) in these parts. However you might still find some nodes where the data is not homogeneous, and therefore the entropy would not be that small.

3.6.2 Decision Tree Classification in Python

What does the fit method do here?
It will simply train the Decision Tree model on X_train and y_train. More precisely, the fit method will collect the data in X_train and y_train, and from that it will do all the successive operations you saw in the Intuition Lecture: first it will select the criterion (which we choose to be the entropy in our code), and then it will split the data into different successive nodes in the direction of entropy reduction, towards more homogeneous nodes. Eventually when all the splits are made, it will be fully trained and ready to predict the outcomes of new observations thanks to the predict() method.

How to plot the graph of a tree in Python?
Here is how you can plot a Decision Tree in Python (just add the following code at the end of your code):

```
# Plotting the tree
# In the terminal enter: pip install pydot2
from sklearn import tree
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydot
dot_data = StringIO()
tree.export_graphviz(classifier,
                     out_file = dot_data,
                     feature_names = ['Age', 'Estimated Salary'],
                     class_names = ['Yes', 'No'],
                     filled = True,
                     rounded = True,
                     special_characters = True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

3.6.3 Decision Tree Classification in R

How were the splits made in R
It made these splits with salaries grouped in three levels, based on entropy and information gain. The splits were made so that the parts resulting from the splits are as much homogeneous as possible.

Why is y_pred a matrix in this model while in all the other model it is a vector?
It's a specificity of the Decision Tree model: the predictions are returned in a matrix of two columns with the probabilities prob(y=0) and prob(y=1).

3.7 Random Forest Classification

3.7.1 Random Forest Classification Intuition

What is the advantage and drawback of Random Forests compared to Decision Trees? Advantage: Random Forests can give you a better predictive power than Decision Trees.
Drawback: Decision Tree will give you more interpretability than Random Forests, because you can plot the graph of a Decision Tree to see the different splits leading to the prediction, as seen in the Intuition Lecture. That's something you can't do with Random Forests.

When to use Random Forest and when to use the other models?
The best answer to that question is: try them all!
Indeed, thanks to the templates it will only take you 10 minutes to try all the models, which is very little compared to the time dedicated to the other parts of a data science project (like Data Preprocessing for example). So just don't be scared to try all the classification models and compare the results (through cross validation which we will see in Part 10). That's we gave you the maximum models in this course for you to have in your toolkit and increase your chance of getting better results.
However then, if you want some shortcuts, here are some rules of thumbs to help you decide which model to use:
First, you need to figure out whether your problem is linear or non linear. You will learn how to do that in Part 10 - Model Selection. Then: If your problem is linear, you should go for a Logistic Regression model or a SVM model, because these are both linear models.
If your problem is non linear, you should go for either K-NN, Kernel SVM, Naive Bayes, Decision Tree or Random Forest. Then which one should you choose among these five? That you will learn in Part 10 - Model Selection. The method consists of using a very relevant technique that evaluates your models performance, called k-Fold Cross Validation, and then picking the model that shows the best results. Feel free to jump directly to Part 10 if you already want to learn how to do that.

3.7.2 Random Forest Regression in Python

How do I know how many trees I should use?
First, I would recommend to choose the number of trees by experimenting. It usually takes less time than we think to figure out a best value by tweaking and tuning your model manually. That's actually what we do in general when we build a Machine Learning model: we do it in several shots, by experimenting several values of hyperparameters like the number of trees. However, also know that in Part 10 we will cover k-Fold Cross Validation and Grid Search, which are powerful techniques that you can use to find the optimal value of a hyperparameter, like here the number of trees.

3.7.3 Random Forest Regression in R

How do we decide how many trees would be enough to get a relatively accurate result?
Same as in Python, you can find a relevant number of tree through experimentation and manual tweaking. You should use enough trees to get a good accuracy, but you shouldn't use too many trees because that could cause overfitting. You will learn how to find the optimal number of trees in the first section of Part 10 - Model Selection. It's done with a technique called Parameter Tuning (Grid Search with k-Fold Cross Validation).

Can we select the most significant variables thanks to the p-value like we did in R before?
You couldn't use the p-value because Random Forests are not linear models, and p-values apply only to linear models. Therefore feature selection is out of the question. But you could do feature extraction, which you will see in Part 9 - Dimensionality Reduction. That you can apply to Random Forests, and it will reduce the number of your features.

How can be reduced overfitting of Random Forests?
You can do that by playing with the penalization and regularization parameters if there are some in the class used to build the model. However the best and most efficient way to reduce overfitting, especially for Random Forests, is to apply k-fold cross validation and optimize a tuning parameter on bootstrapped data. Don't worry we will see all this in depth in Part 10 - Model Selection & Ensemble Learning.

3.8 Evaluating Classification Models Performance

How much should we rely on the confusion matrix?
We can use the confusion matrix to get a sense of how potentially well our model can perform. However we shouldn't limit our evaluation of the performance to the accuracy on one train test split, as it is the case in the confusion matrix. First, there is the variance problem so you should rather apply k-Fold Cross Validation to get a more relevant measure of the accuracy. You will learn how to do that in the first section of Part 10 - Model Selection. Then, there is not only the accuracy that you should consider. You should also consider other metrics like Precision, Recall, and the F1 Score. You will see that in Part 7.

Is cumulative gain curve the same thing as CAP curve?
Yes that's the same, Gain Curve is the name for CAP Curve in a marketing context.

If you target 20000 customers and if 4000 respond, then why does this not have a multiplying affect? So if I target 100000 customers then I should get a response from 50000?
The whole idea here is that , we can maximise the returns by picking out the customers that may purchase the most, such that if p(customer_will_purchase) is higher than some threshold, then we should send an e-mail.
The idea is that you will never get a response from 50,000 since, whatever happens, only 10,000 customers will buy on each offer.
So, what we would like to do is that from our customer base of 1,00,000, pick out those 10,000 customers exactly that will buy for the current offer. If we have a model so good, that it can accurately pin-point everyone who will buy, then we can just e-mail that many people.
There is no multiplicative effect here. Let me explain. Even if we send e-mails to all customers(100k), then only 10k will buy. (90% of the customers is not usually interested in each offer) . Our goal here is to minimise the e-mails we send out by sending emails to the 10k that will buy our offer. So we can use probabilistic models to predict wether or not a customer will buy. If the probability that the customer will purchase is higher than some threshold, we can send them an e-mail. This will increase the chance for each customer to purchase the offer, while keeping the number of e-mails to a minimum.

Should we plot the CAP curve in Python or R?
The Excel template provided in the course is more recommended. First, export your values in Python:

```
classifier = SVC(kernel = 'rbf', probability = 1, random_state = 0)
Y_pred_proba = classifier.predict_proba(X_test)
Y_cap = np.c_[Y_test, Y_pred_proba[:, 1:]]
dataframe = pd.DataFrame(data = Y_cap)
dataframe.to_csv('CAP.csv', sep=';')
```

Then, use this CAP Curve template that will directly give you the CAP curve once you paste your exported values inside this template.
As for R, there are a couple of R packages you can use to calculate a CAP curve: "ROCR" and "MASS".

What are Low/High Bias/Variance in Machine Learning?
Low Bias is when your model predictions are very close to the real values.
High Bias is when your model predictions are far from the real values.
Low Variance: when you run your model several times, the different predictions of your observation points won't vary much.
High Variance: when you run your model several times, the different predictions of your observation points will vary a lot.
What you want to get when you build a model is: Low Bias and Low Variance.

4 Part 4 - Clustering

4.1 K-Means Clustering

4.1.1 K-Means Clustering Intuition

Where can we apply clustering algorithm in real life?
You can apply them for different purposes:
- Market Segmentation,
- Medicine with for example tumor detection,
- Fraud detection,
- to simply identify some clusters of your customers in your company or business.

How does the perpendicular line trick work when $k \geq 3$?
This trick is mainly used only in 2D or 3D spaces. Usually, the Euclidean distance is used on all points for high dimensions to perform the clustering. In very high dimensional data, we could do a trick, which is an "engulfing sphere" i.e. starting at every centroid (k spheres in total), start growing a spherical space (circle in 2D, sphere in 3D, etc), outward radially. Until the spheres intersect, everything the sphere engulfs belongs to the same cluster. All the other points can be assigned to clusters by calculating the perpendicular distance between the centroids.

How does the elbow method work when more than 2 features are involved?
It is the same but we use the Euclidean distance in n dimensions to compute the WCSS (p and q are two observation points, and their coordinates $p_1, ..., p_n, q_1, ..., q_n$ are the features of these observation points):

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

Is there an alternative way to the Elbow method to find the ultimate number of clusters?
Sure, an alternative way would be through Parameter Tuning with Grid Search, which you will see in Part 10 - Model Selection.

4.1.2 K-Means Clustering in Python

What is exactly the parameter 'max_iter' in the KMeans() class?
When we apply k-Means, we jump back to the previous step many times (see the Intuition Lectures). Each time we do that, that is an iteration. And max_iter is the maximum number of these iterations.

Why do we not consider the age as a parameter?
We don't consider the age for two reasons:
1. I had pre-checked it had not impact on the dependent variable.
2. I wanted to have two independent variables in the end, so that we can visualize the results in two dimensions (since one independent variable corresponds to one dimension).

How to get statistics for each of the clusters?
The simplest way to get some statistical information of the clusters is to use the cluster_centers_ attribute of the KMeans class. It will give an array with the coordinates of each cluster, that is the mean of each feature column of each cluster. Then you can also get some info about the observation points by using the labels_ attribute. You can call for these attributes this way:

```
kmeans.cluster_centers_
kmeans.labels_
```

Then, you could get some useful values like the number of elements in each cluster, the mean of the salaries or spending scores in each cluster, etc. The best way to do this is to put your five clusters into 5 variables this way:

```
Cluster_0 = X[y_kmeans == 0]
Cluster_1 = X[y_kmeans == 1]
Cluster_2 = X[y_kmeans == 2]
Cluster_3 = X[y_kmeans == 3]
Cluster_4 = X[y_kmeans == 4]
```

Then you get whatever you want with these variables that are exactly the clusters.
For example if you are interested in having the count of observations for each cluster, well that you can now see it directly in Variable Explorer by looking at the number of rows for each cluster. Or you can do this:

```
len(Cluster_0)
len(Cluster_1)
len(Cluster_2)
len(Cluster_3)
len(Cluster_4)
```

Then you can get the mean and the standard deviation of each feature of your cluster this way:

```
Cluster_0[0].mean()
Cluster_1[0].mean()
Cluster_2[0].mean()
Cluster_3[0].mean()
Cluster_4[0].mean()
Cluster_0[0].std()
Cluster_1[0].std()
Cluster_2[0].std()
Cluster_3[0].std()
Cluster_4[0].std()
```

You can also get the percentage of observations by the following simple computations:

```
len(Cluster_0) / 200
len(Cluster_1) / 200
len(Cluster_2) / 200
len(Cluster_3) / 200
len(Cluster_4) / 200
```

Please explain the following line of code in more details:

```
plt.scatter(kmeans.cluster_centers_[:,0],
            kmeans.cluster_centers_[:, 1],
            s = 300,
            c = 'yellow',
            label = 'Centroids')
```

In the scatter() function of the plt module:
- the first argument is the x-coordinate of the cluster centers (the centroids),
- the second argument is the y-coordinate of the cluster centers (the centroids),
- the third argument s is the size of the centroid points in the plot,
- the fourth argument is the colour of the centroid points in the plot,
- the fifth argument is the label appearing in the legend corresponding to the centroids.

4.1.3 K-Means Clustering in R

How do I not fall into the random initialization trap in R?
R takes care of not falling into random initialization trap behind the scenes. However you have some other great packages in R where you can use explicitly k-means++, it's the KMeans_rcpp package.

Why haven't we used all the features like the age?
I didn't use them because I pre-checked that they had no impact on the dependent variable. Besides I wanted to have two independent variables so that we could visualize the results in two dimensions (because one independent variable corresponds to one dimension).

What is the role of nstart here?
Because K-Means Clustering starts with k randomly chosen centroids, a different solution can be obtained each time the function is invoked. Using the set.seed() function guarantees that the results are reproducible. Additionally, this clustering approach can be sensitive to the initial selection of centroids. The kmeans() function has an nstart option that attempts multiple initial configurations and reports on the best one. For example, adding nstart = 25 generates 25 initial configurations. This approach is often recommended.

How can I plot the results without the scaling and with the names of the clusters?
Here is an implementation of such a plot:

```
plot(x = dataset[,1],
     y = dataset[,2],
     col = y_kmeans,
     pch = 19,
     xlim = c(from=min(dataset[,1]), to=max(dataset[,1]+30)),
     xlab = "Annual Income", ylab="Spending Score")
clusters = c("Careless", "Standard", "Sensible", "Target", "Careful")
legend('bottomright', legend=clusters, col=1:5, pch=19, horiz=F)
```

4.2 Hierarchical Clustering

4.2.1 Hierarchical Clustering Intuition

What is the point of Hierarchical Clustering if it always leads to one cluster per observation point?
The main point of Hierarchical Clustering is to make the dendrogram, because you need to start with one single cluster, then work your way down to see the different combinations of clusters until having a number of clusters equal to the number of observations. And it's the dendrogram itself that allows to find the best clustering configuration.

When you are comparing the distance between two clusters or a cluster and a point, how exactly is it measured? Are you taking the centroid in the cluster and measuring the distance?
Exactly, the metric is the euclidean distance between the centroid of the first cluster and the point, (or the centroid of the other cluster for the distance between two clusters).

Do we also need to perform feature scaling for Hierarchical Clustering?
Yes because the equations of the clustering problems involve the Euclidean Distance. Anytime the model equations involve the Euclidean Distance, you should apply feature scaling.

4.2.2 Hierarchical Clustering in Python

Should we use the dendrogram or the elbow method to find that optimal number of clusters?
You should use both (it's faster to try both than you think thanks to the templates), just to double check that optimal number. However if you really only have time for one, I would recommend the elbow method. The dendrogram is not always the easiest way to find the optimal number of clusters. But with the elbow method it's very easy, since the elbow is most of the time very obvious to spot.

Could you expand on how the clusters are formed through the ward method of the AgglomerativeClustering() Python class?
Hierarchical clustering is a general family of clustering algorithms that build nested clusters by merging or splitting them successively. This hierarchy of clusters is represented as a tree (or dendrogram). The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample. The AgglomerativeClustering() class performs a hierarchical clustering using a bottom up approach: each observation starts in its own cluster, and clusters are successively merged together. The ward method minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.

I thought that dendrograms were like the memory of the HC algorithm. If so, why do we first make the dendrogram and then apply the agglomerative clustering? Does that mean that we execute twice the clustering algorithm?
The only purpose of making first the dendrogram is to have an idea of the optimal number of clusters. It's like the memory or the history of the HC algorithm. But through that history/memory we can see the optimal number of clusters. And that's what we do in this step. Then when we apply agglomerative clustering, we can input the optimal number of clusters that we found thanks to the dendrogram.

What is 'affinity' in the AgglomerativeClustering() class?
'affinity' refers to the distance used in the K-Means algorithm, which is the Euclidean distance (the most classic one, the geometrical one). It refers to how the HC algorithm defines (finds) the closest centroids to each point. And then what goes along with it is the ward method used to build the clusters, meaning that we use the Within Cluster Sum of Squares as a criterion to gather and form our clusters.

Why don't we simply implement some code that automatically finds the optimal number of clusters, instead of selecting it manually or visually?
Because the choice of the optimal number of clusters is also influenced by the business problem (goals and constraints). For example, some business problems have a minimum or maximum number of clusters, or a min/max number of elements per cluster. Such constraints require us to have several options, and the best way to figure out the best options are by looking at the graphs (dendrogram or elbow).

4.2.3 Hierarchical Clustering in R

Why did we not apply feature scaling in R?
Because the function we use in R automatically takes care of feature scaling itself.

How can I visualize the clusters with the original scale?
If you want to look at the clusters with original scales you can use the following commands:

```
clusters= 1:5
plot(x=dataset[,1], y=dataset[,2],
     col=y_hc, pch=19, xlim=c(from=min(dataset[,1]), to=max(dataset[,1]+20)))
legend('bottomright', inset=0.01, legend=clusters, col=1:5, pch=19, horiz=F)
```

5 Part 5 - Association Rule Learning

5.1 Apriori

5.1.1 Apriori Intuition

What are the three essential relations between the support, confidence and lift?
Given two movies M_1 and M_2, here are the three essential relations to remember:
Relation between the support and the confidence:

$$\text{confidence}(M_1 \to M_2) = \frac{\text{support}(M_1, M_2)}{\text{support}(M_1)}$$

Relation between the lift and the support:

$$\text{lift}(M_1 \to M_2) = \frac{\text{support}(M_1, M_2)}{\text{support}(M_1) \times \text{support}(M_2)}$$

Relation between the lift and the confidence (consequence of the two previous equations):

$$\text{lift}(M_1 \to M_2) = \frac{\text{confidence}(M_1 \to M_2)}{\text{support}(M_2)}$$

Are the confidence and lift symmetrical functions?
Given the three equations of the previous question, we can easily see that:
Confidence is non symmetrical:

$$\text{confidence}(M_1 \to M_2) \neq \text{confidence}(M_2 \to M_1)$$

Lift is symmetrical:

$$\text{lift}(M_1 \to M_2) = \text{lift}(M_2 \to M_1)$$

5.1.2 Apriori in Python

Important question: is R better than Python for Association Rule Learning?
Yes, absolutely. The R package is much more optimized and easy to use. But for R haters we made sure to provide a Python solution. However if you don't mind, we highly recommend to go with R.

I have trouble finding the rules in Python. How can I see them more explicitly?
Sorry about that, the package version changed after I recorded the Lecture. But you can see them by just adding the following at the end of your Python code (in a new section right after the "Visualising the results" section):

```
the_rules = []
for result in results:
    the_rules.append({'rule': ','.join(result.items),
                      'support':result.support,
                      'confidence':result.ordered_statistics[0].confidence,
                      'lift':result.ordered_statistics[0].lift})
df = pd.DataFrame(the_rules, columns = ['rule', 'support', 'confidence', 'lift'])
```

After execution of this code, you should see a DataFrame called 'df' appearing in the 'Variable Explorer' with all the required information.

5.1.3 Apriori in R

Where does the 3 come from in the support calculation?
I picked 3 because I considered the products that are bought at least 3 times a day. Because less than 3 times a day seemed to be not relevant enough to include them in the analysis.

Why should the support and confidence be minimum?
It is because we want our rules to be significant and reflect real trend, not something really rare which happens once in a few years, but by accident got in our records.

Why isn't a rule like mineral water (lhs) and whole wheat pasta (rhs) not showing up at the top of the list if rules?
It's because mineral water can go with anything, so the association rule between mineral water and whole wheat pasta is not the most relevant. Think of it this way: if you buy mineral water, do you want to buy whole wheat pasta as first choice?

How would you measure how good the rules here are? You mentioned how Amazon and Netflix use more advanced algorithms but how have they come to the conclusion that their algorithms are superior?
In Association Rule Learning, you simply measure that with the lift. The higher are the lifts, the better your rules are. Amazon and Netflix work with lots of algorithms including Association Rules, Recommender Systems and other advanced Machine Learning models. They evaluate their models through performance metrics and experimenting.

In real time scenario what is the ideal time period we should consider to make a good Market Basket Analysis model? And should it be done on each store separately or region wise?
One month is a good time period. However you could also consider 3-6 months to normalize the seasonality effect, or you can run the same model every month, which I would rather do to catch the specificities of each month (season, tourism rate, etc.).
Then Market Basket Analysis should be done on each store separately, since it depends on the customer behaviors within a specific neighborhood. Basically customers might behave differently across different neighborhoods.

5.2 Eclat

5.2.1 Eclat Intuition

When should we use Eclat rather than Apriori?
The only advantage of Eclat compared to Apriori is that it is simpler and faster to use. However if you need to run a deep analysis of your market basket, then you should definitely go for Apriori.

5.2.2 Eclat in Python

Could you provide an Eclat implementation in Python?
Credit to Marcello Morchio, a student of this course who kindly shared his implementation:

```
# Eclat, by Marcello Morchio, December 4th 2017

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Importing the dataset
dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None )
transactions = [[str(dataset.values[i,j]) for j in range(0, dataset.shape[1])
                 if str(dataset.values[i,j]) != 'nan'] for i in range(0, dataset.shape[0])]

# Generate a list of unique items in the transactions
items = list()
for t in transactions:
    for x in t:
        if(not x in items):
            items.append(x)

# Generate a list of pairs of items with relevant support value
# [[(item_a, item_b) , support_value]]
# support_value is initialized to 0 for all pairs
eclat = list()
for i in range(0, len(items)):
    for j in range(i+1, len(items)):
        eclat.append([(items[i],items[j]),0])

# Compute support value for each pair by looking for transactions with both items
for p in eclat:
    for t in transactions:
        if(p[0][0] in t) and (p[0][1] in t):
            p[1] += 1
    p[1] = p[1]/len(transactions)

# Converts eclat in sorted DataFrame to be visualized in variable explorer
eclatDataFrame = pd.DataFrame(eclat,
columns = ['rule','support']).sort_values(by = 'support', ascending = False)
```

5.2.3 Eclat in R

It seems that Mineral Water is skewing the results of the associations because it is the most frequent purchase, so would it be unreasonable to simply exclude it from the data and re-build the model without it?
Yes that could be relevant. However one would need to check the share of revenue coming from mineral water. If the share is high, maybe it's not a bad idea to place it next to its best association product after all.

We found 5 duplicates. Shouldn't we remove them?
Absolutely not. It could mean that 5 people by accident bought the same, so this records represent different customers and then they should be saved.

What is the density explained at 03:23 in the Eclat R lecture?
density is a term that can have different meanings in different contexts. In this particular case the density is the proportion of cells with items (non zero values in the matrix) over the total number of cells in the matrix.

6 Part 6 - Reinforcement Learning

6.1 Upper Confidence Bound (UCB)

6.1.1 UCB Intuition

Could you please explain in greater details what a distribution is? What are on the x-axis and y-axis?
Let us assume that we have an experiment with ad clicking 100 times, and compute what is the frequency of ad clicking. Then we repeat it again for another 100 times. And again for another 100 times. Hence we obtain many frequencies. If we repeat such frequency computations many times, for example 500 times, we can plot a histogram of these frequencies. By the Central Limit Theorem it will be bell-shaped, and its mean will be the mean of all frequencies we obtained over the experiment. Therefore in conclusion, on the x-axis we have the different possible values of these frequencies, and on the y-axis we have the number of times we obtained each frequency over the experiment.

In the first Intuition Lecture, could you please explain why D5 (in orange) is the best distribution? Why is it not D3 (in pink)?
In this situation, 0 is loss, and 1 is gain, or win. The D5 is the best because it is skewed so we will have average outcomes close to 1, meaning that there we have more wins, or gains. And actually all casino machines nowadays are carefully programmed to have distribution like D1 or D3. But it is a good concrete example.

6.1.2 UCB in Python

Why does a single round can have multiple 1s for different ads?
Each round corresponds to a user connecting to a web page, and the user can only see one ad when he/she connects to the web page, the ad that is being shown to him/her. If he/she clicks on the ad, the reward is 1, otherwise it's 0. And there can be multiple ones because there are several ads that the user would love to click on. But only a crystal ball would tell us which ads the user would click on. And the dataset is exactly that crystal ball (we are doing a simulation here).

Could you please explain in more details what we do with the rewards at lines 33 & 34?
Let's explain clearly:
- In line 33 we are just getting the reward at the specific round n. So we get a 1 if the user clicked on the ad at this round n, and 0 if the user didn't.
- In line 34 we are getting the sum of all the rewards up to round n. So this sum is incremented by 1 if the user clicked on the ad at this round n, and stays the same if the user didn't.

Does the UCB strategy really come into effect during the first rounds?
Not at the beginning, first we must have some first insights of the users response to the ads. That is just the beginning of the strategy. In real world for the first ten users connecting to the webpage, you would show ad 1 to the first user, ad 2 to the second user, ad 3 to the third user,..., ad 10 to the 10th user. Then the algorithm starts.

What was the purpose of this algorithm? Why couldn't I just count which type of ad converted the most and then used that?
Because at each round there is a cost (the cost of putting one ad on the page). So basically the purpose of UCB is not only to maximize the revenue but also to minimize the cost. And if you just count which type of ad converted the most, that would require you to experiment a lot of rounds and therefore the cost would be high and definitely not minimized.

6.1.3 UCB in R

How to get this kind of dataset?
In real life, you could do this by data streaming, using Spark or Hive, meaning you would get real time data. Otherwise if you want evaluate your Reinforcement Learning models, you can simulate a dataset like the one we use in the tutorials.

What is exactly the 'number of selections'?
'number of selections' is the number of times the ad was displayed to a user.

I don't understand why there is no ad selection by the algorithm in the first 10 rounds. The instructor said there is no strategy at the beginning so the first round will display Ad 1, 2nd round Ad 2, 3rd round Ad 3, and so on.. So why is there no selection by UCB?
Because at this point we know nothing about ads and we cannot apply our formula because it involves division by how many times an ad was shown. We may not divide by 0.

In Python the winning ad was ad no. 4 and in R it was ad no. 5. However the same Ads_CTR_Optimisation.csv file was used. Why we are getting two different results using the same logic?
Because indexes in Python start from 0 and indexes in R start from 1. Hence Python and R both get the same ad.

Could you please provide a R implementation of the UCB regret curve?

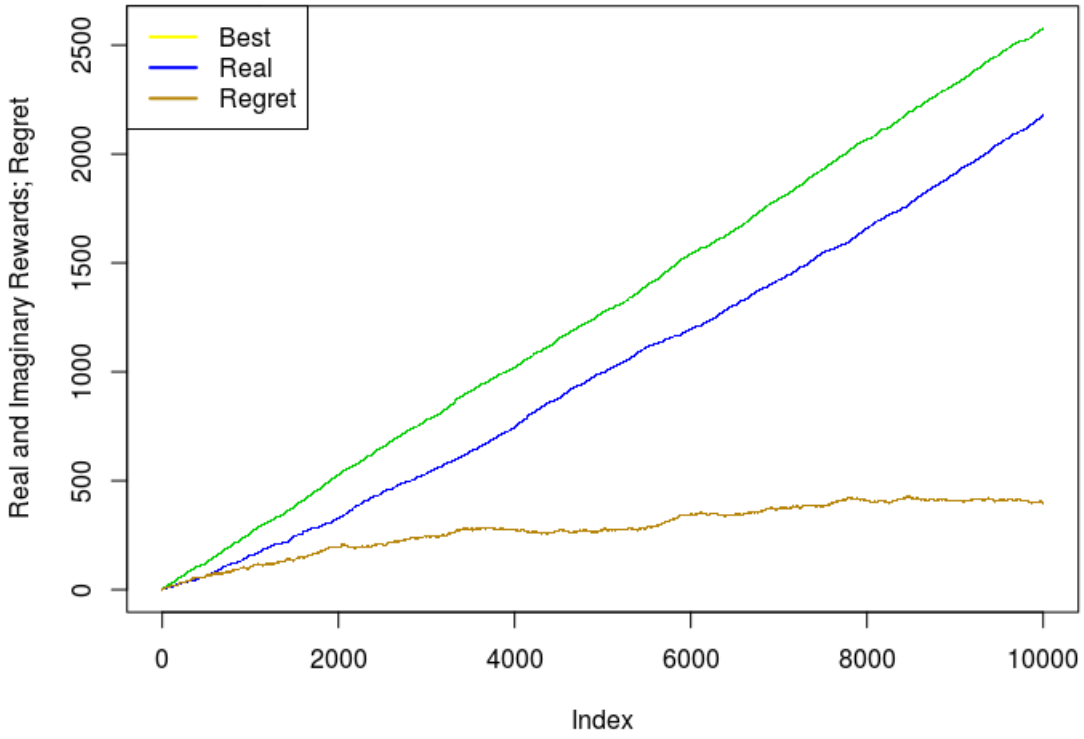
```
# Importing the dataset
dataset = read.csv('Ads_CTR_Optimisation.csv')

# Implementing UCB
N = 10000
d = 10
ads_selected = integer(0)
numbers_of_selections = integer(d)
sums_of_rewards = integer(d)
total_reward = 0
rewards_at_each_step=integer(0)
best_selection=(rowSums(dataset)==0) # my addition
best_rewards_at_each_step=integer(0) # my addition
for (n in 1:N) {
  ad = 0
  max_upper_bound = 0
  for (i in 1:d) {
    if (numbers_of_selections[i] > 0) {
      average_reward = sums_of_rewards[i] / numbers_of_selections[i]
      delta_i = sqrt(3/2 * log(n) / numbers_of_selections[i])
      upper_bound = average_reward + delta_i
    } else {
        upper_bound = 1e400
    }
    if (upper_bound > max_upper_bound) {
      max_upper_bound = upper_bound
      ad = i
    }
  }
```

```
  ads_selected = append(ads_selected, ad)
  numbers_of_selections[ad] = numbers_of_selections[ad] + 1
  reward = dataset[n, ad]
  sums_of_rewards[ad] = sums_of_rewards[ad] + reward
  total_reward = total_reward + reward
  rewards_at_each_step=c(rewards_at_each_step, total_reward) #my addition
  best_rewards_at_each_step[n]=sum(best_selection[1:n]) # my addition
}

# Regret curve
plot(best_rewards_at_each_step, pch='.', col=3,
    main="Real and Imaginary Rewards; Regret",
    ylab="Reward Numbers")
points(rewards_at_each_step, pch=".", col=4)
points((best_rewards_at_each_step-rewards_at_each_step), pch='.', col="darkgoldenrod")
legend('topleft', legend=c("Best", "Real", "Regret"),
    col=c(7,4, "darkgoldenrod"),  horiz=F, lty=1, lwd=2)
```

And we obtain the following plot:



6.2 Thompson Sampling

6.2.1 Thompson Sampling Intuition

Why is the yellow mark the best choice, and not the green mark?
The yellow mark is the best choice because it is the furthest from the origin on the x-axis, which therefore means that it has the highest estimated return.

How is Thompson Sampling better than UCB?
Thompson Sampling is better than UCB in terms of convergence of the regret. The regret is the difference between the optimal reward and the reward you accumulate with your algorithm. Thompson Sampling shows a better regret curve than UCB in my experience. Also, the fact that UCB is deterministic as opposed to Thompson Sampling being stochastic, helps making Thompson Sampling outperform UCB. Besides you will see in the practical sections that Thompson Sampling finds the best ad faster and with more certainty than UCB.

I don't understand how Thompson Sampling can accept delayed feedback. Please explain.
When doing Thompson Sampling, we can still perform updates in our algorithm (like making new guesses for the distributions with existing data, sampling from the guessed distribution, etc) while we are waiting for the results of an experiment in the real world. This would not hinder our algorithm from working. This is why it can accept delayed feedback.

What are further examples of Thompson Sampling applications?
Another potential application of Multi-armed bandits (MAB) can be the online testing of algorithms.
For example, let's suppose you are running an e-commerce website and you have at your disposal several Machine Learning algorithms to provide recommendations to users (of whatever the website is selling), but you don't know which algorithm leads to the best recommendations.
You could consider your problem as a MAB problem and define each Machine Learning algorithm as an "arm": at each round when one user requests a recommendation, one arm (i.e. one of the algorithms) will be selected to make the recommendations, and you will receive a reward. In this case, you could define your reward in various ways, a simple example is "1" if the user clicks/buys an item and "0" otherwise. Eventually your bandit algorithm will converge and end up always choosing the algorithm which is the most efficient at providing recommendations. This is a good way to find the most suitable model in an online problem.
Another example coming to my mind is finding the best clinical treatment for patients: each possible treatment could be considered as an "arm", and a simple way to define the reward would be a number between 0 (the treatment has no effect at all) and 1 (the patient is cured perfectly).
In this case, the goal is to find as quickly as possible the best treatment while minimizing the cumulative regret (which is equivalent to say you want to avoid as much as possible selecting "bad" or even sub-optimal treatments during the process).

6.2.2 Thompson Sampling in Python

Where can I find some great resource on the Beta distribution?
The best is the following: Beta Distribution

The histogram shows 4 as the best ad to select. The highest bar is at 4 on the x axis. Why is it being said as 5?
It's just because indexes in Python start from 0. So the ad of index 4 it ad number 5.

I am curious as to how one would apply Thompson Sampling proactively when running this theoretical ad campaign. Would you iterate the program over each round (i.e. each time all adds were presented to the user)?
First, a data engineer makes a pipeline for reading the data from website and reacting to it in real time. Then a web visit triggers a response to recompute our parameters and to choose an ad for the next time.

6.2.3 Thompson Sampling in R

Could you please explain in your own words the difference between the Bernoulli and Beta distributions?
The Bernoulli distribution is applied to discrete probabilities because it gives the probability of success of an outcome, whereas the Beta distribution is applied to dense (continuous) probabilities, using a dense probability function. That's the main difference. Here are some excellent resources:
Bernoulli distribution
Beta distribution

Could you please provide a great source on Bayesian Inference?
Here is a great one: Bayesien Inference

How is Thomson Sampling heuristic quickly able to find that 5th advertisement is the best one in comparison to the Upper Confidence Bound heuristic?
It is hard to explain the reason theoretically, that would require to do research and write a long mathematical proof. But intuitively, it could be because UCB is based on optimistic assumptions whereas Thompson Sampling is based on relevant probabilities through the Bayesian approach.

How to plot the Thompson Sampling Regret Curve in R?
Check out the last question in the UCB in R section. Basically you simply need to add the following at the end of your code:

```
rewards_at_each_step=c(rewards_at_each_step, total_reward)
  best_rewards_at_each_step[n]=sum(best_selection[1:n])
}
plot(best_rewards_at_each_step, pch='.', col=3,
     main="Real and Imaginary Rewards; Regret",
     ylab="Reward Numbers")
points(rewards_at_each_step, pch=".", col=4)
points((best_rewards_at_each_step-rewards_at_each_step), pch='.', col="darkgoldenrod")
legend('topleft', legend=c("Best", "Real", "Regret"),
       col=c(7,4, "darkgoldenrod"),  horiz=F, lty=1, lwd=2)
```

7 Part 7 - Natural Language Processing

7.1 Natural Language Processing Intuition

Why does the Bag of Words model replace all the capital letters by the lower cases?
Many computer languages, in particular Python and R, treat capitals and lower case letters as completely different symbols. So if we won't convert everything to lower cases then we are to take into account that some words may contain capitalized letters and include additional code for them. Much more simpler to replace all capitals by lower cases.

What is sparsity?
sparsity occurs when you have lots and lots of zeros in your matrix (therefore called a sparse matrix). So when we reduce sparsity, that means we reduce the proportion of zeros in the matrix.

7.2 Natural Language Processing in Python

Could you please explain what 'quoting = 3' means?
quoting = 3 will ignore the double quotes in the texts. The number of observations in the dataset didn't match the number of texts which was due to a splitting anomaly with the double quotes. The potential problem is that if you have double quotes in a review, this review can accidentally be separated and considered as another review with no outcome. So to make sure we avoid this kind of situation we ignore the double quotes by using quoting = 3.

How can we download all the NLTK material in one shot?
To download all the material in NLTK in one shot you need to open a terminal and run:

```
pip install -U nltk
```

Could you please explain what is PorterStemmer()?
PorterStemmer() applies stemming to the words in your texts. So for example, "loved" will become "love" after the stemming.

How to do NLP in French or other languages?
Check if there are some NLP classes specific to your own language, as it is the case in French for example:

```
from nltk.stem.snowball import FrenchStemmer
stemmer = FrenchStemmer()
```

and for the stopwords, include this in your code:

```
set(stopwords.words('french'))
```

Why did we change the reviews from lists of words back to strings?
It is done because the fit_transform() method from the CountVectorizer() class requires strings to work.

What is the difference between "TF-IDF" and "CountVectorizer"?
The difference is that TF-IDF applies normalization, unlike CountVectorizer. Otherwise it is the same, if you input the parameter 'normalize = None', then it's very similar.
To use TF-IDF, it's the same as for the other NLP techniques, you just import the class you want (say TfidfVectorizer from 'sklearn.feature_extraction.text'), then you create an object of this class while inputing the parameters you want to preprocess your texts.

Why do we delete 'NOT', given that "Crust is good" is not the same as "Crust is not good"?
It also works the other way: "Crust is bad" is not the same as "Crust is not bad". Hence, this is not usually taken into account. Hence we believe/hope that on average, the number of misconceptions caused by this would be less in total.

After we are done with cleaning and getting our bag of words model, why are we doing classification?
We are doing classification on the reviews to predict the outcome of new ones, exactly like sentiment analysis. Only to make these predictions in the best conditions, we need to apply the bag of words model first. That will prepare the data in a correct format for it to be fitted to the classifier.

What model seems to be the best solution to the Homework Challenge?
Definitely Naive Bayes, which by the way is usually recommended first when doing Natural Language Processing.

7.3 Natural Language Processing in R

Why do we remove punctuation from the text? At first they look like they don't contain information. But why we won't let classification algorithm to decide?
It is usually a trade-off between dimensionality and information. We remove punctuation to make the dataset more manageable with less dimensions.
Moreover, the punctuation doesn't really carry as much discriminatory information as words in usual classification processes.
Through experience, cross-validation has pointed towards removing punctuation performing better.
However yes, we could let the classification algorithm decide. Maybe some datasets perform better with punctuation but as a general-rule, punctuation is better to be avoided.

"crust not good" is being transformed to "crust good" after removing irrelevant words. The review is changing to positive from negative. Can you clarify how this will impact learning and prediction?
The word "not" is not as helpful as it looks. Consider the following 2 reviews:
"I did not order their pasta. Their bread was good."
"I did order their pasta. Their bread was not good."
From the "bag of words" approach they are the same, although their meaning is very different.
Usually when people want to take "not" in the account they would replace the following word with antonym and remove the negation.

Is there any way or resources through which I can take a look at the list of the non-relevant words?
You can see them by entering into R the following command:

```
library(tm)
stopwords("english")
```

What would be another technique than confusion matrix and CAP curve to test the NLP model?
Another and best technique is k-Fold Cross Validation, which you will see in Part 10 - Model Selection.

8 Part 8 - Deep Learning

8.1 Artificial Neural Networks

8.1.1 Artificial Neural Networks Intuition

Into what category does ANN fall? Supervised, Unsupervised, Reinforced Learning?
ANNs can be all 3. Depends on how you implement the model. Examples:
Supervised Learning: CNNs classifying images in imagenet.
Unsupervised Learning: Boltzmann Machines, AutoEncoders, GANs, DC-GANS, VAE, SOMs, etc.
Reinforcement: Deep Convolutional Q-Learning that plays videogames from pixel input, AlphaGO, etc. This branch is called "Deep Reinforcement Learning" and belongs to Artificial Intelligence".

How does the Neural Network figure out what the optimal weights are. When does it stop its learning?
It figures out the optimal weights based on an optimization algorithm employed to minimize the loss function w.r.t. all the input datapoints. It stops when the training loss goes very close to 0. Besides we usually pick a certain number of epochs which is the number of times the neural network is trained. After the last epoch, the training is over. There is also what we call "Early Stopping" which is a technique that stops the training once we get a too small loss reduction on a validation set over the epochs. In that case, the training stops before the last epoch.

Why should the cost function be reduced?
The cost function is a numerical estimate of how wrong our neural network predictions are. If we reduce the cost function, it means we are reducing the mistakes made by the neural network, in-turn making it predict more accurately.
How is the weight calculated for each neuron?
At the beginning of the training, the weights are initialized close to zero. Then, over each of many iterations (or epochs), the weights are updated through gradient descent, in the direction that decreases the most the loss error (or cost function) between the predictions and the targets.

Could you please recap on the process of each training iteration?
After the weights are initialized randomly with values close to zero, the training goes over many iterations (the number of iterations, also called the number of epochs, is usually decided in the implementation). Here is the process of each iteration:

1. Forward Propagation: the input is forward propagated inside the neural network which at the end returns the prediction.
2. We compare that prediction to the target (the real value which we have because we are dealing with the training set) and we compute the loss error between the prediction and the target.
3. That loss error is back-propagated inside the neural network.
4. Through Gradient Descent or Stochastic Gradient Descent, the weights are updated in the directions that reduce the most the loss error (since in fact the loss error is a cost function of the weights).
5. We thus get new weights, ready to make a new prediction at the next iteration. Then the same whole process goes again, until the loss error stops reducing (early stopping) or until the training reaches the last epoch.

8.1.2 Artificial Neural Networks in Python

Why would an ANN be a better model/solution than just using one of the different models we learned in Part 3?
It's not necessarily the best solution for all problems. For example, if your problem is linear, then this is definitely not the best solution since you would be much more efficient with a Linear Regression model. However if your problem is non linear and complex, then ANN can be a better model/solution than the models in Part 3. In Part 10 - Model Selection you will learn how to evaluate that.

Why are we using Sequential and Dense?
Basically it's very simple:
- Sequential is used to initialize the Deep Learning model as a sequence of layers (as opposed to a computational graph).
- Dense is used to add one layer of neurons in the neural network.

I get some warnings when I execute the code section where I add the layers with the Dense() function. How can I fix that?
You simply need to replace:

```
classifier.add(Dense(output_dim = 6,
                     init = 'uniform',
                     activation = 'relu',
                     input_dim = 11))
```

by:

```
classifier.add(Dense(units = 6,
                     kernel_initializer = "uniform",
                     activation = "relu",
                     input_dim = 11))
```

Basically some parameters names changed in the API: 'output_dim' became 'units', 'init' became 'kernel_initializer' and 'nb_epoch' became 'epochs'. It happens frequently in libraries.

How can we decide the number of hidden layers?
There is no rule of thumb and generally more hidden layers can give you higher accuracy but can also give you overfitting so you need to be cautious. The key is intuition and experience. You can also experiment with Parameter Tuning which you will see in Part 10 - Model Selection.

What does the rectifier activation function do?
Since Deep Learning is used to solve non linear problems, the models need to be non linear. And the use of the rectifier function is to actually make it non linear. By applying it you are breaking the linearity between the output neurons and the input neurons.

What do I need to change if I have a non-binary output, e.g. win , lose and draw?
In that case you would need to create three dummy variables for your dependent variable:
(1,0,0) → win (0,1,0) → lose (0,0,1) → draw And therefore you need to make the following change:

```
output_dim = 3 # now with the new API it is: units = 3
activation = 'softmax'
loss = 'categorical_crossentropy'
```

nb_epoch has been updated?
Yes, in your code you need to replace 'nb_epoch' by 'epochs'.

How to build the same neural network for Regression?
First you need to import:

```
KerasRegressor
```

Then it's almost the same as an ANN for Classification. The main difference is the compile function, where of course we need to change the loss function and take a loss function for Regression like the Mean Squared Error, and remove the accuracy metric:

```
# Initialising the ANN
regressor = Sequential()

# Adding the input layer and the first hidden layer
regressor.add(Dense(units = 6,
                    kernel_initializer = 'uniform',
                    activation = 'relu',
                    input_dim = 11))

# Adding the second hidden layer
regressor.add(Dense(units = 6,
                    kernel_initializer = 'uniform',
                    activation = 'relu'))

# Adding the output layer
regressor.add(Dense(units = 1,
                    kernel_initializer = 'uniform'))

# Compiling the ANN
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the ANN to the Training set
regressor.fit(X_train,
              y_train,
              batch_size = 10,
              epochs = 100)
```

Where do all the hyperparameters come from? How did we choose the number of layers, the number of neurons in each layer, and all the other parameters?
Through a lot of research and experimentation. There is no rule of thumb as to choose such numbers. Usually you can find some ready to use neural networks architectures online which prove to get good results. But you can experiment by tuning your model manually with other parameter values. And lastly, you can use some parameter tuning techniques like Grid Search with k-Fold Cross Validation to find the optimal values. You will learn how to do that in Part 10 - Model Selection.

How can I improve the accuracy of the ANN?
This is the million dollar question. Data scientist have been trying to figure out this answer for a while. Usually ANN models take a good amount of tuning, this is usually in the form of changing number of hidden layers, changing formulas, and normalizing data. Usually these things will get you the farthest.

8.1.3 Artificial Neural Networks in R

Why do we use 'as.numeric(factor())'?
Because to build the neural network we will then use the H2O package which expects the inputs as numerical factors (or numeric categorical variables).

If I choose hidden = $c(a_1, a_2, a_3..a_n)$, does this mean there are n hidden layers in the model, and $a_1, a_2, ..., a_n$ are the numbers of nodes in each hidden layer?
Yes, that's correct!

Is the number of nodes in each hidden layer always the same? Or we can specify different nodes in each hidden layer? How do we choose these numbers?
No, you may play with it and see if you can get better results. We advise to watch your task manager though, to see if your computer memory is sufficient to handle so many layers and their nodes. You can choose them by either experimenting manually (trial and error) or through Parameter Tuning with Grid Search (see Part 10 - Model Selection).

In Python we applied the Rectifier Function to the Hidden Layers, but the Sigmoid Function to the Outcome Layer. How to choose the same options with R?
Using H2O in R, the Rectifier Function was applied to each layer. Here there is no option of choosing different activation functions for different layers.

Python or R for Deep Learning?
Python. Without a doubt. Python is used by all the best Deep Learning scientists today and it has amazing libraries (Tensorflow, Keras, PyTorch) developed for powerful applications (Image Classification, Computer Vision, Artificial Intelligence, ChatBots, Machine Translation, etc.).

8.2 Convolutional Neural Networks

8.2.1 Convolutional Neural Networks Intuition

What are the differences between the CNN and the ANN? Is the CNN applied to image classification only?
ANN means neural networks in general. So a CNN is a ANN. CNNs are not only used for Image Processing, they can also be applied to Text like Text Comprehension.
Down to how much the image size will be reduced for the feature detector?
In the practical section it will be reduced down to 64 by 64 dimensions.

What is the purpose of the feature maps?
We are building feature maps with each convolutional filter, meaning we are making features that help us classify the object. The example shown in the Intuition Lecture is like a one-dimensional edge detection filter. That is one feature map. So we want our model to "activate" in a feature map only where there is an edge. We will have several feature maps like this which when all put together will help us identify the object. This is helped by removing the black or the negative values.

What is the purpose of the ReLU?
The biggest reason why we use ReLU is because we want to increase the non-linearity in our image. And ReLU acts as a function which breaks up linearity. And the reason why we want to break up linearity in our network is because images themselves are highly non-linear. Indeed they have a lot of non-linear elements like the transitions between pixels.

Why does Max-Pooling consider only 4 values to take a maximum from and not 2 or 8 values? Also in convolution how is the feature detector formed?
Because a maximum is taken from a spot on the image which is represented by a 2x2 square on our image. Therefore it covers 4 pixels.

After flattening, are we going to get one long vector for all pooled layers or a vector for each pooled layer?
It will be one long vector gathering all the pooled layers.

8.2.2 Convolutional Neural Networks in Python

How much images are required to train a good model? Is there any rule of thumb for this decision?
10,000 is a good number. No rule of thumbs, just the more you have, the better chance you'll have to get a good accuracy.

Could you please explain the numbers 0, 1, 2, 3 and 4 in the feature maps?
When the 3x3 feature detector is covering one part of the input image, it gets:
0 if there is no 1 in common between the 3x3 subtable of the input image and the 3x3 feature detector,
1 if there is one 1 in common between the 3x3 subtable of the input image and the 3x3 feature detector,
2 if there is two 1 in common between the 3x3 subtable of the input image and the 3x3 feature detector,
3 if there is three 1 in common between the 3x3 subtable of the input image and the 3x3 feature detector,
4 if there is four 1 in common between the 3x3 subtable of the input image and the 3x3 feature detector.

Could you please recap the forward propagation of the input images happening inside the CNN by describing in a few words the classes we use in Python?
Sure, here is the process with the essential descriptions of the classes used:
Sequential is first used to specify we introduce a sequence of layers, as opposed to a computational graph.
Convolution2D is then used to add the convolutional layer.
MaxPooling2D is then used to apply Max Pooling to the input images.
Flatten is then used to flatten the pooled images.
Dense is used to add the output layer with softmax.

How to make a single prediction whether a specific picture contains a cat or a dog?
Inside the dataset folder you need to create a separate additional folder (let's call it "single_prediction") containing the image (let's call it "cat_or_dog.jpg") you want to predict and run the following code:

```
import numpy as np
from keras.preprocessing import image as image_utils
test_image = image_utils.load_img('dataset/single_prediction/cat_or_dog.jpg',
                                  target_size = (64, 64))
test_image = image_utils.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict_on_batch(test_image)
training_set.class_indices
if result[0][0] == 1:
    prediction = 'dog'
else:
    prediction = 'cat'
```

9 Part 9 - Dimensionality Reduction

9.1 Principal Component Analysis (PCA)

9.1.1 PCA Intuition

What is the true purpose of PCA?
The true purpose is mainly to decrease the complexity of the model. It is to simplify the model while keeping relevance and performance. Sometimes you can have datasets with hundreds of features so in that case you just want to extract much fewer independent variables that explain the most the variance.

What is the difference between PCA and Factor Analysis?
Principal component analysis involves extracting linear composites of observed variables.
Factor analysis is based on a formal model predicting observed variables from theoretical latent factors.
PCA is meant to maximize the total variance to look for distinguishable patterns, and Factor analysis looks to maximize the shared variance for latent constructs or variables.

Should I apply PCA if my dataset has categorical variables?
You could try PCA, but I would be really careful, because categorical values can have high variances by default and will usually be unstable to matrix inversion.
Apply PCA and do cross validation to see if it can generalize better than the actual data. If it does, then PCA is good for your model. (Your training matrix is numerically stable). However, I am certain that in most cases, PCA does not work well in datasets that only contain categorical data. Vanilla PCA is designed based on capturing the covariance in continuous variables. There are other data reduction methods you can try to compress the data like multiple correspondence analysis and categorical PCA etc.

What is the best extra resource on PCA?
Check out this video that has an amazing explanation of PCA and studies it in more depth.

9.1.2 PCA in Python

What does the fit_transform do here? Why do we apply fit_transform to the training set, and only transform to the test set?
In the fit_transform method there is fit and transform.
The fit part is used to analyze the data on which we apply the object (getting the eigen values and the eigen vectors of the covariance matrix, etc.) in order to get the required information to apply the PCA transformation, that is, extracting some top features that explain the most the variance.
Then once the object gets these informations thanks to the fit method, the transform part is used to apply the PCA transformation.
And since the test set and the training set have very similar structures, we don't need to create a new object that we fit to the test set and then use to transform the test set, we can directly use the object already created and fitted to the training set, to transform the test set.

How do you find out which ones of the independent variables are the top 2 components?
PCA is a feature extraction technique so the components are not one ones of the original independent variables. These are new ones, like some sort of transformations of the original ones.
It's only with feature selection that you end up with ones of the original independent variables, like with Backward Elimination.

Is it better to use Feature Extraction or Feature Selection, or both? If both, in which order?
Feature Extraction and Feature Selection are two great dimensionality reduction techniques, and therefore you should always consider both.
What I recommend is doing first Feature Selection to only keep the relevant features, and then apply Feature Extraction on these selected relevant features to reduce even more the dimensionality of your dataset while keeping enough variance.

How much total variance ratio do we need to use? Is there any threshold for good total variance ratio?
Generally a good threshold is 50%. But 60% is more recommended.

Is it more common to use exactly 2 independent variables to build a classifier, or do people typically use more than that?
In general people just extract a number of independent variables that explain a sufficient proportion of the variance (typically 60%). So it's not always two. It can be more. And if it's two that is great because then you can visualize better.

Is there a Python method or attribute that can help provide the underlying components and signs of the two principal components PC1 and PC2?
Sure you can access them with the components_ attributes:

```
components = pca.components_
```

9.1.3 PCA in R

If I wanted to see the actual values for all the columns, how would I unscale the data which was feature scaled?
Say that your scaled vector was y_train, then to unscale your result y_pred do the following:

```
y_pred_unscaled = y_pred*attr(y_train,'scaled:scale') + attr(y_train, 'scaled:center')
```

How can I see the principal components in R?
You can run the following code:

```
pca$rotation
```

We got an accuracy of 100%, shouldn't we be worried of overfitting?
If you look at the data, you'll see that it's almost perfectly separable by the 3 lines drawn, which means that the separability is a characteristic of the data rather than an overfitting problem.
If you had something like 50 lines and 100% accuracy (i.e., each section would capture precisely a small amount of points, guaranteeing that they are rightly classified), than it would probably be an overfitting issue, but here that is clearly not the case.
Besides we obtained 100% accuracy only on the test set. There were some mis-classified points in the training set. And overfitting is rather the opposite: an almost perfect accuracy on the training set and a poor one on the test set.

How can we know which are the two variables taken for plotting?
The two variables are not among your original independent variables, since they were extracted (Feature Extraction), as opposed to being selected (Feature Selection). The two new variables are the directions where there is the most variance, that is the directions where the data is most spread out.

9.2 Linear Discriminant Analysis (LDA)

9.2.1 LDA Intuition

Could you please explain in a more simpler way the difference between PCA and LDA?
A simple way of viewing the difference between PCA and LDA is that PCA treats the entire data set as a whole while LDA attempts to model the differences between classes within the data. Also, PCA extracts some components that explain the most the variance, while LDA extracts some components that maximize class separability.

Feature Selection or Feature Extraction?
You would rather choose feature selection if you want to keep all the interpretation of your problem, your dataset and your model results. But if you don't care about the interpretation and only car about getting accurate predictions, then you can try both, separately or together, and compare the performance results. So yes feature selection and feature extraction can be applied simultaneously in a given problem.

Can we use LDA for Regression?
LDA is Linear Discriminant Analysis. It is a generalization of Fisher's linear discriminant, a method used in statistics, pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification. However, for regression, we have to use ANOVA, a variation of LDA. LDA is also closely related to principal component analysis (PCA) and factor analysis in that they both look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data. PCA on the other hand does not take into account any difference in class, and factor analysis builds the feature combinations based on differences rather than similarities. Discriminant analysis is also different from factor analysis in that it is not an interdependence technique: a distinction between independent variables and dependent variables (also called criterion variables) must be made. LDA works when the measurements made on independent variables for each observation are continuous quantities. When dealing with categorical independent variables, the equivalent technique is discriminant correspondence analysis.

9.2.2 LDA in Python

Which independent variables are found after applying LDA?
The two independent variables that you see, indexed by 0 and 1, are new independent variables that are not among your 12 original independent variables. These are totally new independent variables that were extracted through LDA, and that's why we call LDA Feature Extraction, as opposed to Feature Selection where you keep some of your original independent variables.

How to decide the LDA n_component parameter in order to find the most accurate result?
You can run:

```
LDA(n\_components = None)
```

and it should give you automatically the ideal n_components.

How can I get the two Linear Discriminants LD1 and LD2 in Python?
You can get them by running the following line of code:

```
lda.scalings_
```

9.2.3 LDA in R

How can I get the two Linear Discriminants LD1 and LD2 in R?
You can get them by running the following line of code:

```
lda$scaling
```

I don't quite get why the R version of LDA picked 2 independent variables automatically?
Because in R, when you have k classes, k-1 is equal to the number of linear discriminants that you get.

Why are we getting matrix in LDA but got a dataframe in PCA, even though the procedure is similar?
This is just due to a difference in the format that the function provides as an output. These procedures are similar regardless of the format that is output.

9.3 Kernel PCA

9.3.1 Kernel PCA Intuition

Should Kernel PCA be used to convert non-linearly separable data into linearly separable data?
That's right, but you don't need to use Kernel PCA with a non linear classifier since the data will be linearly separable after applying Kernel PCA, and therefore a linear classifier will be sufficient.

When should we use PCA vs Kernel PCA?
You should start with PCA. Then if you get poor results, try Kernel PCA.

9.3.2 Kernel PCA in Python

How do I know if my data is linearly separable or not?
A good trick is to train a Logistic Regression model on it first. If you get a really good accuracy, it should be (almost) linearly separable.

Is there a huge difference and what is better to use between Kernel PCA + SVM vs PCA + Kernel SVM?
Yes there is a difference.
Use Kernel PCA + SVM when you can transform your data into a non-linear low dimensional manifold where the points are separable.
Use PCA + Kernel SVM when you need to transform your data through a linear transformation into a low dimensional manifold, using these points to be transformed into a non-linear space where they are separable.

How do we decide which kernel is best for Kernel PCA?
The RBF Kernel is a great kernel, and is the best option in general. But the best way to figure out what kernel you need to apply is to do some Parameter Tuning with Grid Search and k-Fold Cross Validation. We will see that in Part 10 - Model Selection.

9.3.3 Kernel PCA in R

How can I get the Principal Components of Kernel PCA in R?
You simply need to execute the following line of code:

```
attr(kpca, "pcv")
```

10 Part 10 - Model Selection & Boosting

10.1 k-Fold Cross Validation

10.1.1 k-Fold Cross Validation Intuition

What is low/high bias/variance?
These concepts are important to understand k-Fold Cross Validation:
Low Bias is when your model predictions are very close to the real values.
High Bias is when your model predictions are far from the real values.
Low Variance: when you run your model several times, the different predictions of your observation points won't vary much.
High Variance: when you run your model several times, the different predictions of your observation points will vary a lot.

Does k-Fold Cross Validation improve the model or is it just a method of validation?
k-Fold Cross Validation is used to evaluate your model. It doesn't necessarily improve your model, but improves your understanding of the model. However you can use it to improve your model by combining it with some Parameter Tuning techniques like Grid Search (next section).

What is the difference between a parameter and a hyperparameter?
Hyper parameters and parameters are very similar but not the exact same thing. A parameter is a configurable variable that is internal to a model whose value can be estimated from the data.
A hyperparameter is a configurable value external to a model whose value cannot be determined by the data, and that we are trying to optimize (find the optimal value) through Parameter Tuning techniques like Random Search or Grid Search.

What is a good/best value of k to choose when performing k-Fold Cross Validation?
We strongly recommend 10.

10.1.2 k-Fold Cross Validation in Python

I was wondering if we needed to split the data into X_train, y_train and X_test, y_test to apply the resampling method(k-Fold) or can we apply it directly on X and y?
You can do both. The good method is:

1. Split your dataset into a training set and a test set.
2. Perform k-fold cross validation on the training set.
3. Make the final evaluation of your selected model on the test set.

But you can also perform k-fold Cross-Validation on the whole dataset (X, y).

What does this Standard Deviation tell us exactly?
The Standard Deviation of the model's accuracy simply shows the variance of the model accuracy is 6%. This means the model can vary about 6%, which means that if I run my model on new data and get an accuracy of 86%, I know that this is like within 80-92% accuracy. Bias and accuracy sometimes don't have an obvious relationship, but most of the time you can spot some bias in the validation or testing of your model when it does not preform properly on new data.

How to calculate the F1 score, Recall or Precision from k-Fold Cross Validation?
You can use sklearn's metrics library for this. Here is a link.

Why only Kernel SVM?
You can apply k-Fold Cross Validation to any Machine Learning model, and would be very smart to do this for every model you create. Kernel SVM was just an example here to explain how to apply k-Fold Cross Validation.

10.1.3 k-Fold Cross Validation in R

We used the confusion matrix to determine the accuracy for each fold. What would you use for regression models? R-squared?
No we would rather use the MSE (Mean Squared Error), that measures the sum of the squared differences between the actual results and the predictions.

The k-Fold Cross Validation example in R creates 10 different models using the Training set. How can the mean of 10 different models evaluate the original model? Do we even need the original model?
Through k-Fold Cross Validation, we need to estimate the "unbiased error" on the dataset from this model. The mean is a good estimate of how the model will perform on the dataset. Then once we are convinced that the average error is acceptable, we train the same model on all of the dataset.

In R we created folds from the training set. Shouldn't we create them for the entire data set? Why just the training set?
We only use the training set to make our model. The test set is considered as something we don't have with us yet. (Even though we do). So we use the training set to make the best model and then evaluate its performance on the test set. If we tune the performance on the test set, then we have a model biased to performing well on the test-set, which is only a very small instance of the whole data. We need our model to perform well for the data we haven't seen yet as well.

10.2 Grid Search

10.2.1 Grid Search in Python

I cannot import 'GridSearchCV' on my computer. Any other options for this package?
It depends on the system and package version, but try to replace:

```
from sklearn.model_selection import GridSearchCV
```

by:

```
from sklearn.cross_validation import GridSearchCV
```

What is this C penalty parameter?
The C penalty parameter is a regularization parameter that can allow you to do two things:
- reduce overfitting,
- override outliers.
It is equal to $\frac{1}{\lambda}$ in where λ is the classic regularization parameter used in Ridge Regression, Lasso, Elastic Net.

Can Grid Search be extended to ANN as well?
Yes, Grid search is possible in ANN as well. We cover it in our "Deep Learning A-Z" course.

How do we know which values we should test in the Grid Search?
A good start is to take default values and experiment with values around them. For example, the default value of the penalty parameter C is 10, so some relevant values to try would be 1, 10 and 100.

In this Python section, we are tuning parameters in Grid Search to yield the optimal accuracy in the training set with k-Fold Cross Validation? How can we assure that these parameters will be the most optimal when working with the test set?
These parameters will be the most optimal when working with the test set because they proved to be optimal on ten different test sets (the ten test set folds in the cross-validation process).

10.2.2 Grid Search in R

How to use C and sigma value calculated in Grid Search?
Once you have found the optimal C and sigma value, you can put these values into your SVM model arguments and obtain a optimal accuracy for your model.

What are the advantages and disadvantages of using caret in R?
Advantages of caret:

1. When doing Grid Search you don't have to pick values of hyperparameters to try yourself, it will find some optimal ones for you.
2. It has a lot of other functions besides being a convenient Machine Learning method wrapper.
3. It has its own splitting into training and test sets and it contains a function which creates splitting with time series.
4. It has tools for data cleaning as well, and they all have very efficient implementation.
5. If you use it to fit a model, it can bootstrap your data to improve your model.

Disadvantages:

1. It is a big package and takes a chunk of your memory.
2. The bootstrapping feature is very compute intensive.

In conclusion, caret is an amazing R package for Model Selection.

What is this Kappa value that we see in R?
That is the Cohen's kappa. It is a metric used to evaluate the performance of a classifier and also it helps evaluating classifiers among themselves as well. For more info have a look at this good reference here.

When doing Grid Search in R can't we know if the data is linearly separable or not, just like in Python?
Sure, just like in Python you can see which Kernel SVM gives you the best accuracy using Grid Search with caret. If the Kernel is linear, then your data is probably linearly separable. And if the Kernel is not linear, your data is probably not linearly separable.

10.3 XGBoost

10.3.1 XGBoost Intuition

Could you please provide a good source that explains how XGBoost works?
We recommend to be introduced first to Gradient Boosting algorigthms by going through this great source. Then we recommend to go here to understand XGBoost.

10.3.2 XGBoost in Python

I have issues installing XGBoost. How can I install it easily?
When the Lecture was made, there was no conda installation command. But now there is and it is much simpler to install the XGBoost package. You simply need to enter the following command inside a terminal (or anaconda prompt for Windows users):

```
conda install -c conda-forge xgboost
```

Can XGBoost be applied to both Classification and Regression in Python?
Absolutely. For Classification, you use the XGBClassifier class. And for Regression, you use the XGBRegressor class.

10.3.3 XGBoost in R

In the k-Fold Cross Validation code section, line 35, shouldn't it be 'training_fold' instead of 'training_set'?
Indeed! Thanks for noticing that. Of course it's the training fold. Our apologies for the mistake. Therefore the right code section is:

```
classifier = xgboost(data = as.matrix(training_fold[-11]),
                     label = training_fold$Exited,
                     nrounds = 10)
```

Can XGBoost be applied to both Classification and Regression in R?
Absolutely. And you use the same xgboost() function for both Classification and Regression.