Questions:

1. How do supervised learning algorithms solve regression and classification problems?

Supervised learning is a subset of machine learning that learns a function that maps an input to an output based on available examples of input-output pairs. In supervised learning a function is inferred from labeled training data (set of training examples) and can be then used to map new examples.

There are different algorithms used in supervised machine learning including linear and logistic regression, multi-class classification, decision trees and support vector machines. All supervised machine learning tasks can be broadly divided into two subgroups: regression and classification. Regression tasks include estimating and predicting continuous variables, classification on the other hand assigns observations into discrete categories.

In any case, supervised learning required that the data used to train the algorithm is labeled with correct answers. For example, in order to train an algorithm to classify handwritten digits we need pictures of handwritten digits along with labels containing the correct number each image represents. The algorithm then will learn the relationship between the images and the numbers they represent and will be able to apply this learned relationship to classify previously unseen unlabeled images. In case of handwritten image recognition, we might need thousands of training images for the algorithm to become reasonably accurate. In simpler cases, for example classifying among several dog breeds based on their height, weight, coat type and color, a much smaller training data set might be sufficient.

Regression predicts a continuous target variable based on one or more input variables. For example, we can build a regression model to predict a person's yearly income based on year of education, years of work experience, occupation, and location. Again, the data is split into a training data set that includes labels (annual income) and a test data set (no labels indicating annual income). The regression model results in a mathematical formula estimating annual income based on input values that can be generalized to situations that the model has not seen before (test data).

2. What packages (in R, Python…) perform supervised learning?

Because of its open source development paradigm, R has many packages that are designed to perform supervised learning tasks. It might not be possible to compile an all inclusive list of those packages. Overall, CRAN package repository (https://cran.r-project.org/web/packages/) currently features 12936 available packages. A subpages dedicated to machine learning (https://cran.r-project.org/web/views/MachineLearning.html) lists at least 101 package that among others include supervised and unsupervised machine learning tools in various combinations. One of the most popular packages is ***caret***, which is a package that attempts to integrate several algorithm specific packages. Another attempt to summarize features and integrate several machine learning algorithms is the ***mlr*** package (https://cran.r-project.org/web/packages/mlr/vignettes/mlr.html). The mlr package has a goal of providing unified interface for machine learning tasks as classification, regression, cluster analysis and survival analysis in R. Among popular task-specific packages used for supervised learning in R I would list:

- E1071 -used to implement Naïve Bayes (conditional probability) models, Support Vector Machines;
- Rpart – used to build classification and regression models;
- RandomForest – used to create a large number of decision trees.

3. What measures of quality of the learning algorithm might you expect to see?

The main measure if quality of a learning algorithm is its accuracy of predicted results. Depending on a task, it can be accuracy of classification or how close the values predicted using a regression model are to the actual observed results. In other words, the quality of an algorithm is defined by its accuracy/percentage of errors.

In practice, accuracy can be evaluated using various statistical methods. In case of a classification algorithm, the simplest way if to use a confusion matrix, or a contingency table comparing predicted classification results to the actual classification using testing dataset. Its result will allow us to calculate percentage of wrongly classified observations. In case of regression analysis, statistical indicators, such as root mean square of errors, for example, can be used to evaluate the accuracy of predicted values and several parametric and non-parametric test can be applied (for example, ANOVA, for linear regression and non-parametric tests for non-linear models).

However, in practice some additional factors might need to be taken into consideration when evaluating quality of learning algorithms. For example, a classification model can demonstrate acceptable results if we count just overall errors, but those errors might be unevenly distributed between classes. It might be potentially a sign of a biased model. Regression models have to be analyzed by looking at either absolute vs. relative to magnitude errors or squared errors.

After completing the exercise (using K-nearest neighbors algorithm for classifying Iris, I decided to try to build a model detecting spam text messages using Naïve Bayes algorithm and text processing capabilities offered by ***quanteda*** package. Since it was my first experience with this package, in building this classifier, I adapted some steps suggested in the following tutorial https://blog.paperspace.com/intro-to-datascience/) I used a publicly available text messaging dataset from the Machine Learning Repository at UCI found at https://archive.ics.uci.edu/ml/machine-learning-databases/00228/ . This data set contains 5574 real text messages of which 13.4% are spam and 86.6% are legitimate messages (ham).

First, I prepared the environment and loaded the **quanteda** package using the following code:

```
> #M Supervised Learning
>
> rm(list=ls()) #Clear the environment
> setwd("YOUR_PATH") #Set working directory for the assignment
> getwd() #Check working directory
[1] "YOUR_PATH"
```

```
> ########################################################################
> ######Using Supervised Learning for Spam Detection#####################
> ######Based on A Gentle Introduction to Data Classification with R######
> ######https://blog.paperspace.com/intro-to-datascience/#################
> ########################################################################
```

```
> library(quanteda)
```

Next, I loaded previously obtained at https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection data into a table using the following code :

```
> #Load Data into a table
> raw.data <- read.table("SMSSpamCollection", header = FALSE, sep="\t", quote="", stringsAsFactor
s = FALSE)
```

For convenience, I added column names:

```
> #Add column names
> names(raw.data) <- c("Label", "Text")
```

To make sure that the data was loaded correctly, I checked the structure of the data table and its content:

```
> #Check the structure and content of the data table
> summary(raw.data)
    Label               Text
 Length:5574        Length:5574
 Class :character   Class :character
 Mode  :character   Mode  :character
> table(raw.data$Label)

 ham spam
4827  747
```

The data was loaded correctly. Since the data set was originally compiled from three different sources, I used sample() command to randomize it:

```
> #Randomizse data using sample() command
> set.seed(1912)
> raw.data <- raw.data[sample(nrow(raw.data)),]
> View(raw.data) #Check data
```

Preview of the results is below:

Filter

	Label	Text
814	spam	Congratulations ur awarded either Â£500 of CD gift vou...
1436	ham	I asked sen to come chennai and search for job.
438	ham	Ask g or iouri, I've told the story like ten times already
3833	ham	Let's pool our money together and buy a bunch of lotto...
3668	ham	I'm turning off my phone. My moms telling everyone I h...
4953	ham	hi baby im sat on the bloody bus at the mo and i wont b...
3480	ham	I can ask around but there's not a lot in terms of mids u...
521	ham	Usually the person is unconscious that's in children but ...
733	ham	No he didn't. Spring is coming early yay!
4165	ham	I told that am coming on wednesday.
1225	ham	Rofl betta invest in some anti aging products
2694	spam	Urgent Urgent! We have 800 FREE flights to Europe to g...
267	ham	Same. Wana plan a trip sometme then
783	ham	Hmmm ... I thought we said 2 hours slave, not 3 ... You ar...

In order to proceed with the modeling, I constructed a corpus object using the text field in the raw data table and attached labels to each row using docvars() command:

```
> ###Construct corpus object
> sms.corpus <- corpus(raw.data$Text) #Construct corpus using text field
> docvars(sms.corpus) <- raw.data$Label #attach Label field as a document variable using docvars(
) command
```

The text used in analysis required only minimal pre-processing as, unlike full text documents and even emails, it is common to SMS messages to use abbreviations, punctuation and other symbols. Deleting these features could actually decrease a model's precision. So, I used dfm() command to load data into a document term frequency matrix (DFM):

```
> ####Pre-processing sms data, wieght word counts
> sms.dfm <- dfm(sms.corpus, tolower = TRUE) #convert to lower case
> sms.dfm <-dfm_trim(sms.dfm, min_docfreq = 3)
> sms.dfm <- dfm_weight(sms.dfm)
```

Since the initial data was randomized, we can just split it into two parts for training (80%) and testing purposes (20%) without running risk of misrepresenting either spam or ham messages.

First, I split the initial data into two subsets:

```
> #Split the data set into training and testing subsets
> #80% training and 20% testing data
> #since the initial set was randomised we can just split
```

```
> #observations [1:4459] and [4460:5574]
> sms.raw.train <-raw.data[1:4459,]#Training part of the raw dataset
> sms.raw.test <-raw.data[4460:nrow(raw.data),] #Testing part of the raw data
```

Then, split the preprocessed data:

```
> #Split DFM into training and testing parts
> sms.dfm.train <- sms.dfm[1:4459,] #training part DFM
> sms.dfm.test <- sms.dfm[4460:nrow(raw.data),] #Testing part DFM
```

For the main step of fitting a Naïve Bayes classification model I used a function textmodel_nb that is included in quanteda package. The model sms.classifier was fitted using training DFM and a vector of associated labels from the raw training data subset:

```
> #Use a Naive Bayes classification model to classify the text messages
> #use textmodel_nb command from quanteda package
>
> ###Fit the model
> #Use training DFM and a vector of associated labels - sms.raw.train$Label
> sms.classifier <- textmodel_nb(sms.dfm.train, sms.raw.train$Label)
```

In the next step, I used this fitted model to classify previously unseen messages from the testing part of the dataset using predict() command and displayed the result using table():

```
###Use sms.classifier for predictions on test data
> sms.predictions <-predict(sms.classifier, newdata = sms.dfm.test)
> #Display prediction results vs. actuual labels
> table(sms.predictions, sms.raw.test$Label)

sms.predictions ham spam
           ham  947    9
           spam   9  150
```

```
> ###Fit the model
> #Use trainging DFM and a vector of associated labels - sms.raw.train$Label
> sms.classifier <- textmodel_nb(sms.dfm.train, sms.raw.train$Label)
> ###Use sms.classifier for predictions on test data
> sms.predictions <-predict(sms.classifier, newdata = sms.dfm.test)
> #Display prediction results vs. actuual labels
> table(sms.predictions, sms.raw.test$Label)

sms.predictions ham spam
           ham  947    9
           spam   9  150
>
```

Conclusions:

The above results show that the model appropriately classified 947 ham messages as ham, and in 9 cases incorrectly labeled legitimate messages as spam. Also 9 spam messages were incorrectly classified as ham, however in 150 cases spam messages were appropriately labeled as spam.

Overall accuracy of the model can be evaluated as correctly classified messages divided by all classified messages ((947 +150)/1115) which results in 98.39% Or, overall less than 2 percent of messages are classified incorrectly. For ham messages the accuracy was 99.06% and for spam messages the accuracy was a little lower – 94.34%. I was a surprisingly good result meaning that out of 100 spam messages received only about 5 spam messages are likely to get through a spam filter.