Syllabus :: Module -5 (Regression Models- Understand Level)

Building linear models - model fitting, Predict values using models, Analyzing the fit, Refining the model, Regression- types, Unusual observation and corrective measures, Comparison of models, Generalized linear models - Logistic Regression, Poisson Regression, Nonlinear least squares

[CO 5 Comprehend regression modeling using R (Cognitive Knowledge level: Understand)]

Overview of Textbook Contents Aligned with the Syllabus [Module 5]

Based on the syllabus, Module 5 covers "Regression Models" with the following topics:

- Building linear models - model fitting
- Predict values using models
- Analyzing the fit
- Refining the model
- Regression - types
- Unusual observation and corrective measures
- Comparison of models
- Generalized linear models - Logistic Regression, Poisson Regression
- Nonlinear least squares

- **Building linear models - model fitting**:
  - The chapter starts with an **"Example: A Simple Linear Model"**, illustrating how to fit a linear model.
  - It then discusses **"Fitting a Model"** , including the use of the `lm` function.
  - The section **"Helper Functions for Specifying the Model"** provides tools for defining the model formula.
- **Predict values using models**:

    - While not explicitly a separate section, the textbook discusses how to use the fitted model, which implicitly involves predicting values. The examples shown would naturally lead to understanding how predictions are made.
    - The syllabus also mentions "Predict values using models" as part of Module 5's teaching plan.
- **Analyzing the fit**:
    - The chapter includes **"Getting Information About a Model"** , detailing how to extract and interpret information about the fitted model, such as coefficients, standard errors, and R-squared.
- **Refining the model**:
    - The textbook has a section on **"Refining the Model"** , discussing techniques to improve the model based on the analysis of its fit.
- **Regression - types**:
    - The chapter explicitly covers various types of regression beyond simple linear regression. It includes sections on:
        - **"Robust and Resistant Regression"**
        - **"Subset Selection and Shrinkage Methods"** , such as Stepwise Variable Selection, Ridge Regression, Lasso and Least Angle Regression, elasticnet.
        - **"Principal Components Regression and Partial Least Squares Regression"**
        - **"Generalized Linear Models"** , which includes discussions on Logistic Regression and Poisson Regression.
        - **"Nonlinear Regression"**
    - The syllabus also lists "Regression- types of regression" in its teaching plan.
- **Unusual observation and corrective measures**:
    - While not a dedicated section with that exact title in the provided excerpts, the textbook discusses aspects related to this under **"Assumptions of Least Squares Regression"** , as violations of these assumptions can lead to unusual observations. Understanding these assumptions is crucial for identifying and addressing potential issues.
    - The syllabus includes "Unusual observations and corrective measures" as separate points in its teaching plan.
- **Comparison of models**:
    - The textbook implicitly touches on model comparison by presenting various regression techniques and discussing their applicability. However, a dedicated section on formal model comparison methods isn't evident in the provided excerpts.
- **Generalized linear models - Logistic Regression, Poisson Regression**:
    - The chapter has a significant section on **"Generalized Linear Models"** which specifically covers:
        - **"Logistic Regression"**

        - **"Poisson Regression"**
    - These align directly with the syllabus's requirements.
- **Nonlinear least squares**:
    - The textbook includes a section on **"Nonlinear Regression"** , which would likely cover nonlinear least squares as a method for fitting such models.

In addition to these core regression topics, Chapter 20 also touches upon related concepts such as **"Survival Models"** and includes a section on **"Machine Learning Algorithms for Regression"** featuring techniques like Regression Trees, Bagging, Boosting, Random Forests, Neural Networks, and Support Vector Machines. While these might go beyond the immediate scope of the listed syllabus topics, they provide a broader context for regression modeling. The textbook also uses the `predict` function in the context of linear models , which directly relates to predicting values using models as mentioned in the syllabus.

Building linear models - model fitting

Building linear models and model fitting in R primarily involves using the `lm()` function. Module 5 of your syllabus specifically covers "Building linear models - model fitting", and it is allocated one lecture hour. Chapter 20 of the textbook, "R in a Nutshell," is dedicated to "Regression Models" and includes a section on "Fitting a Model".

Key Concepts and Steps in Building Linear Models - Model Fitting:

1. **Linear Regression Assumption:** A linear regression assumes a linear relationship between a continuous response variable (also called the dependent variable) and one or more predictor variables (also called stimulus or independent variables). The model assumes that the response variable *y* can be estimated as a linear function of the predictors *x1, x2, ..., xn*:
    - $y = c0 + c1x1 + c2x2 + ... + cnxn + \varepsilon$
    - where *c0* is the intercept, *c1, c2, ..., cn* are the coefficients for the predictor variables, and ε is the error term.
2. **Formula Specification:** In R, you describe the relationship between the response and predictor variables using a **formula object**. The basic syntax is `response ~ predictor1 + predictor2 + ... + predictorN`.
    - For example, to model `dist` (stopping distance) as a linear function of `speed` (from the `cars` dataset), the formula would be `dist ~ speed`.
    - R has special interpretations for symbols like `+`, `*`, `-`, and `^` in formulas. If you need to use these symbols literally in your model formula, you can use the `identity()`

function `I()` to protect the expression. For instance, to include only the product of `a` and `b`, you would use `lm(y ~ I(a*b))`.
    - The `poly()` function provides a convenient way to specify polynomial terms in a model. For example, `poly(x, degree = 3)` would include terms for x, x^2, and x^3.
3. **Using the `lm()` Function:** The primary function in R for fitting linear models using ordinary least squares is `lm()`. The basic syntax is `lm(formula, data)`.

    - **`formula`**: This argument takes the formula object that specifies the relationship between the variables.
    - **`data`**: This argument specifies the data frame, list, or environment where the variables in the formula can be found.
    - The `lm()` function can take several other arguments to control the model fitting process, including:
        - `subset`: To include only a subset of the observations in the data.
        - `weights`: A numeric vector of weights for each observation.
        - `na.action`: Specifies how to handle missing values (`NA`) in the data. The default is `na.omit`.
        - `method`: The method to use for fitting. The default and typically used method is `"qr"`.
        - `model`, `x`, `y`, `qr`: Logical values specifying whether to return the model frame, model matrix, response vector, and QR decomposition, respectively.
        - `singular.ok`: A logical value indicating whether a singular fit should result in an error.
        - `contrasts`: A list of contrasts to use for factors in the model.
        - `offset`: A vector of offsets to be included in the linear predictor with a constant coefficient of 1.
4. **Model Object:** The `lm()` function **returns an object of class `lm`**. This object contains a lot of information about the fitted model and the fitting operation.

    - When you simply call `lm()` or assign its result to a variable without printing, R doesn't display extensive output. To get information about the fitted model, you need to use **helper functions**.

Example: To fit a linear model to the `cars` dataset where stopping distance (`dist`) is predicted by speed (`speed`), you would use the following code:

```
cars.lm <- lm(dist ~ speed, data = cars)
```

5. This creates an `lm` object named `cars.lm`. To see the results, you would typically use functions like `summary()`, `coefficients()`, etc..

6. **Least Squares Method:** The `lm()` function uses the **least squares method** to estimate the coefficients. This method aims to find the coefficients that **minimize the sum of the squared differences (residuals) between the predicted values (ŷ) and the observed response values (Y)**.

7. **Working with Large Datasets:** For very large datasets, the textbook suggests considering the `biglm()` function instead of `lm()`, as it is designed to use less memory.

In summary, building a linear model and fitting it in R involves defining the relationship between variables using a formula, using the `lm()` function with the formula and data, and understanding that `lm()` returns a model object for which you'll need to use other functions to extract and interpret the results. The process relies on the principle of finding the best linear relationship that minimizes the prediction error based on the provided data.

Extra : **Steps involved in model fitting** **[A revision]**

The steps involved in model fitting, particularly within the context of linear models in R as described in Chapter 20 of "R in a Nutshell", generally involve the following:

1. **Understanding the Goal and Data:** Model fitting begins with the objective of describing a set of data, often using a mathematical formula. This can be to build a predictive model or a descriptive model to better understand the data. It also involves having a continuous response variable and one or more predictor variables.

2. **Specifying the Model Formula:** In R, you define the relationship between the response variable and the predictor variables using a **formula object**. The basic syntax is `response ~ predictor1 + predictor2 + ... + predictorN`. This formula specifies the linear relationship assumed between the variables. Helper functions can aid in specifying more complex model formulas, such as including interactions or polynomial terms using functions like `poly()`.

3. **Using the Model Fitting Function:** In R, for linear models using ordinary least squares, the primary function for model fitting is **`lm()`**. You provide the formula and the data to this function. The basic syntax is `lm(formula, data)`. The `data` argument specifies the data frame, list, or environment where the variables in the formula can be found.

4. **Understanding the `lm()` Function Arguments:** The `lm()` function has several arguments that control how the model is fitted. Key arguments include:

   - `formula`: The model formula.

- `data`: The dataset to be used.
- `subset`: To use only a portion of the data.
- `weights`: To apply weights to different observations.
- `na.action`: To specify how missing values should be handled.
- `method`: The method used for fitting; the default is `"qr"`.
- `contrasts`: To specify contrasts for factor variables.
- `offset`: To include a term with a fixed coefficient of 1.

5. **The Model Object:** The `lm()` function **returns an object of class `lm`**. This object contains various information about the fitted model and the fitting process. To access this information, you use helper functions. Assigning the model object to a name is beneficial for further analysis, understanding, and modifying the model without refitting it each time, which can be time-consuming for complex models and large datasets.

6. **Least Squares Estimation:** The `lm()` function, by default, uses the **least squares method** to estimate the coefficients of the linear model. This method aims to minimize the sum of the squared differences between the observed response values and the values predicted by the linear equation.

In essence, the process of building and fitting a linear model in R involves clearly defining the relationship between variables with a formula and then using the `lm()` function to estimate the model parameters based on the provided data, resulting in a model object that can be further analyzed. For very large datasets, the `biglm()` function might be considered.

Extra : Building linear models in R [A revision]

Building linear models in R involves several key steps, primarily revolving around the use of the **`lm()` function**. The sources indicate that Module 5 of your syllabus covers "Building linear models - model fitting", and Chapter 20 of "R in a Nutshell" provides a detailed explanation of regression models, including how to fit them.

Here's a detailed breakdown of building linear models in R:

1. **Understanding the Linear Model Assumption:** A linear model assumes a **linear relationship between a response variable (dependent variable) and one or more predictor variables (independent variables)**. Mathematically, this relationship can be represented as $y = c0 + c1x1 + c2x2 + ... + cnxn + \varepsilon$, where y is the response, $x1$ to xn are the predictors, $c0$ is the intercept, $c1$ to cn are the coefficients, and ε is the error term.

2. **Specifying the Model using Formulas:** In R, you define the linear relationship using a **formula object**. The syntax is `response ~ predictor1 + predictor2 + ... + predictorN`.

    - For instance, to model stopping distance (`dist`) based on speed (`speed`) from the `cars` dataset, the formula would be `dist ~ speed`.
    - R interprets certain symbols in formulas specially (`+`, `*`, `-`, `^`). To use these symbols literally within a formula, you can use the **`identity()` function, `I()`**. For example, `lm(y ~ I(a*b))` would include only the product of `a` and `b` as a predictor.
    - Polynomial terms can be easily included using the **`poly()` function**. For a cubic polynomial of `x`, you would use `poly(x, degree = 3)`.
3. **Fitting the Model with the `lm()` Function:** The primary function for fitting linear models in R using ordinary least squares is **`lm()`**. The basic syntax is `lm(formula, data)`.

    - **`formula`**: This argument takes the formula object that defines the model.
    - **`data`**: This specifies the data frame, list, or environment containing the variables in the formula.
    - The `lm()` function has several other arguments to control the fitting process:
        - `subset`: To include only a subset of the data.
        - `weights`: A numeric vector of weights for each observation.
        - `na.action`: Specifies how to handle missing values (default is `na.omit`).
        - `method`: The fitting method (default is `"qr"`). R uses **QR decomposition** for efficient calculation of coefficients. You can also specify `method="model.frame"` to just return the model frame.
        - `model`, `x`, `y`, `qr`: Logical values to control whether the model frame, model matrix, response vector, and QR decomposition are returned, respectively.
        - `singular.ok`: Determines if a singular fit should result in an error.
        - `contrasts`: A list of contrasts for factor variables. R provides several built-in contrast options like `"contr.helmert"`, `"contr.sum"`, `"contr.treatment"`, and `"contr.poly"`. The default uses the value from `options("contrasts")`.
        - `offset`: A vector of offsets to be included in the linear predictor with a constant coefficient of 1.
        - `...`: Allows passing additional arguments to lower-level functions like `lm.fit` (for unweighted models) or `lm.wfit` (for weighted models).
4. **The Model Object:** The `lm()` function **returns an object of class `lm`**. This object contains a wealth of information about the fitted model. R, unlike some other statistical software, does not automatically print extensive output when a model is fitted. To access the information, you need to use **helper functions**.

- Assigning the result of `lm()` to a variable (e.g., `cars.lm <- lm(dist ~ speed, data = cars)`) is recommended for easier access to the model and to avoid refitting, which can be computationally expensive for complex models and large datasets.

5. **Getting Information About the Fitted Model:** Several helper functions are used to extract information from the `lm` object. Some common ones include:

   - **`print()`**: Calling the model object's name on the console implicitly uses the `print()` method to show a concise summary of the model call and coefficients.
   - **`summary()`**: Provides a more detailed summary, including coefficients with standard errors, t-values, p-values, R-squared, adjusted R-squared, F-statistic, and residual standard error.
   - **`coefficients()`**: Extracts the estimated coefficients of the model.
   - **`residuals()`**: Returns the residuals (the differences between the observed and predicted values).
   - **`fitted.values()`**: Returns the predicted values based on the model.
   - **`anova()`**: Produces an analysis of variance table, useful for assessing the significance of predictors.
   - **`deviance()`**: Returns the residual sum of squares for a linear model.
   - **`plot()`**: Generates diagnostic plots to assess the model assumptions, such as residuals vs. fitted values, Normal Q-Q plot, Scale-Location plot, Cook's distance plot, and Residuals vs. Leverage plot. The `which` argument can specify which plots to display.

6. **Least Squares Method:** The `lm()` function estimates the model coefficients using the **ordinary least squares (OLS) method**. This method aims to **minimize the sum of the squared differences between the observed response values and the values predicted by the linear model**. The Gauss-Markov theorem states that under certain assumptions, OLS estimators have the smallest variance among all unbiased linear estimators.

7. **Assumptions of Least Squares Regression:** It's important to be aware of the assumptions underlying OLS regression:

   - **Linearity:** The relationship between the response and predictors is linear.
   - **Full rank:** There is no perfect linear relationship (multicollinearity) between the predictor variables.
   - **Independent errors:** The errors are independent of each other.
   - **Homoscedasticity:** The variance of the errors is constant across all levels of the predictors.
   - **Normally distributed errors:** The error terms are normally distributed with a mean of zero.

8. Violations of these assumptions can affect the validity and efficiency of the model. Diagnostic plots from `plot(model)` can help in assessing these assumptions.

9. **Refining the Model:** After fitting an initial model, you might need to refine it. This can involve:
   - **Adding or removing predictor variables** based on their significance and theoretical relevance. The `update()` function can be useful for refitting a model with minor changes to the formula or data.
   - **Transforming variables** to better meet the linearity or homoscedasticity assumptions.
   - **Addressing unusual observations (outliers or influential points)**.
   - **Comparing different models**.
   - **Using subset selection or shrinkage methods** (like stepwise selection, ridge regression, lasso) to potentially improve the model by reducing the impact of less important variables or removing them altogether. The `step()` function performs stepwise variable selection. Functions like `lm.ridge()` (from the `MASS` package) and functions in the `glmnet` package implement ridge and lasso regression.
10. **Working with Large Datasets:** For very large datasets where `lm()` might be memory-intensive, the **`biglm()` function** can be considered, as it is designed to use less memory.

In summary, building linear models in R involves defining the relationship between variables with a formula, using the `lm()` function to fit the model based on the data, and then utilizing various helper functions to examine, interpret, and refine the model. Understanding the assumptions of linear regression is crucial for building reliable models.

Predict values using models

- **The Purpose of Prediction:** Once a model has been built and fitted to a set of data (training data), it can be used to **predict the response variable for new, unseen data**. This is a primary goal of building predictive models.

- **The `predict()` Function:** In R, the primary function used to obtain predictions from a fitted model object is the **`predict()` function**. This function is generic and can be used with various types of models, including linear models (`lm`), generalized linear models (`glm`), and others.

Basic Syntax: The basic syntax for using the `predict()` function is as follows:

predict(object, newdata, ...)

-
   - **`object`**: This argument specifies the **fitted model object** that was returned by the model fitting function (e.g., the `lm` object from `lm()`).

    - **`newdata`**: This argument is an **optional data frame, list, or matrix** containing the values of the predictor variables for which you want to obtain predictions. If `newdata` is not provided, `predict()` might return fitted values for the original data used to build the model. However, it is more common to use `predict()` with new data.
    - **`...`**: This allows for additional arguments that are specific to the type of model being used.
- **Key Arguments of `predict()`:** Several important arguments can be used with the `predict()` function:

    - **`newdata`**: As mentioned, this is crucial for making predictions on data that was not used to train the model. The `newdata` must have **column names that match the predictor variables used in the original model formula**.
    - **`se.fit`**: A logical value indicating whether to return the **standard errors of the predictions**. The default is `FALSE`.
    - **`interval`**: Specifies whether to calculate **prediction or confidence intervals**. Possible values are `"none"` (default), `"confidence"`, and `"prediction"`.
    - **`level`**: Sets the **confidence level** for the intervals, if requested (default is 0.95, meaning 95% intervals).
    - **`type`**: For some models (like `glm`), this argument can specify the **type of prediction** to return (e.g., `"response"` for predicted probabilities in logistic regression, or `"terms"` for the linear predictors). For linear models (`lm`), the default `"response"` typically returns the predicted values of the response variable.
    - **`na.action`**: Specifies how to handle **missing values** in the `newdata`. The default is `na.pass`, which can return `NA` for observations with missing predictor values. You can also choose `na.omit`.
- **Example from the Syllabus:** Course Outcome 5 provides an example where you would use linear regression to predict the weight of a person based on their height using given data. After fitting a linear model with height as the predictor and weight as the response, you would use `predict()` with a new height value to estimate the corresponding weight.

- **Example from "R in a Nutshell":** Chapter 20 mentions predicting values using a model as part of refining the model. It gives an example of a linear model for runs scored in baseball games and explains that the `predict()` function can be used to calculate predicted values using the model object and another data frame.

- **Importance of Assigning Model Objects:** The sources emphasize the importance of assigning the result of the model fitting function (like `lm()`) to a variable name (the `object` in `predict()`). This allows you to easily refer to the fitted model without having to refit it every time you want to make predictions, which can be inefficient for complex models or large datasets.

- **Beyond Linear Models:** While the initial discussion of model fitting in the syllabus focuses on linear models, the `predict()` function is also applicable to other types of models covered later, such as generalized linear models (like logistic and Poisson regression), regression tree models, neural networks, and others discussed in Chapter 20 and potentially Chapter 21 (Classification Models). The specific arguments and output of `predict()` might vary depending on the model type.

In summary, to predict values using models in R:

1. **Fit a model** using a suitable function (e.g., `lm()`, `glm()`, `rpart()`, etc.) and store the resulting model object in a variable.
2. Prepare a **new data frame (`newdata`)** containing the values of the predictor variables for which you want to make predictions. Ensure the column names in `newdata` match the predictors in your model.
3. Use the **`predict()` function**, providing the **fitted model object** and the **`newdata`** as arguments.
4. Examine the **output of `predict()`**, which will contain the predicted values of the response variable for the new data. You can also request standard errors and prediction/confidence intervals using additional arguments.

This process allows you to leverage the relationships learned by your model from the training data to make informed estimates on new, unobserved data.

Analyzing the fit of a model

Analyzing the fit of a model is a crucial step after building and fitting a regression model. It involves assessing how well the model describes the data it was trained on and evaluating its suitability for making predictions.

In "R in a Nutshell," Chapter 20, under the section "Analyzing the fit", provides details on how to perform this analysis, particularly within the context of linear models (`lm`). Here's a breakdown of the key aspects:

- **Using the `summary()` function:** The `summary()` function is a primary tool for analyzing the fit of a linear model. When applied to a fitted model object (e.g., the result of `lm()`), it provides a wealth of information. This includes:
  - **The function call:** This reminds you of the model that was fitted.
  - **Distribution of residuals:** Summary statistics (minimum, first quartile, median, third quartile, maximum) of the residuals, which are the differences between the observed and predicted values. Ideally, residuals should be symmetrically distributed around zero.

- **Coefficients:** For each predictor variable in the model, `summary()` shows the estimated coefficient, its standard error, the t-value (coefficient divided by standard error), and the p-value associated with the t-test. These values help determine the statistical significance of each predictor in the model. Significance codes (e.g., '***', '**', '*') are often used to indicate the level of significance.
- **Residual standard error:** An estimate of the standard deviation of the error term in the regression model. It indicates the average size of the residuals.
- **Multiple R-squared and Adjusted R-squared:** These values represent the proportion of the variance in the response variable that is explained by the model. Multiple R-squared considers all predictors, while Adjusted R-squared accounts for the number of predictors in the model and can help prevent overfitting.
- **F-statistic and p-value:** These are used to test the overall significance of the regression model. The F-statistic assesses whether at least one of the predictor variables has a non-zero coefficient, and the associated p-value indicates the probability of observing such a result if the null hypothesis (all coefficients are zero) were true.

- **Helper functions for getting information about a model:** Besides `summary()`, other functions can be used to extract specific aspects of the model fit:

  - **`formula(model)`:** Returns the formula used to fit the model.
  - **`coef(model)`:** Provides the estimated coefficients of the model.
  - **`residuals(model)`:** Returns the residuals (observed - fitted values).
  - **`fitted(model)` or `fitted.values(model)`:** Returns the predicted values based on the model.
  - **`vcov(model)`:** Returns the variance-covariance matrix of the coefficient estimates.
  - **`confint(model)`:** Computes confidence intervals for the model parameters.
  - **`deviance(model)`:** Returns the residual sum of squares for a linear model.
  - **`AIC(model)`:** Calculates the Akaike Information Criterion, which can be used to compare different models.
  - **`anova(model)`:** Provides analysis of variance tables, which can be useful for understanding the contribution of different predictors.

- **Diagnostic plots using `plot()`:** The `plot()` function, when applied to a fitted `lm` object, generates a set of diagnostic plots that help in assessing the assumptions of the linear regression model and identifying potential problems. The `which` argument can be used to specify which plots to display. Common diagnostic plots include:

  - **Residuals vs Fitted:** This plot should show a random scatter of points with no clear pattern, indicating linearity and homoscedasticity (constant variance of errors).
  - **Normal Q-Q:** This plot compares the distribution of the residuals to a normal distribution. If the residuals are normally distributed, the points should fall approximately along a straight diagonal line.
  - **Scale-Location (Spread vs Fitted):** This plot helps assess the homoscedasticity assumption. A horizontal band of points suggests constant variance.

    - **Cook's Distance:** This plot identifies influential observations that may have a disproportionate impact on the model fit.
    - **Residuals vs Leverage:** This plot helps identify both outliers (large residuals) and high-leverage points (observations with unusual predictor values) and can highlight influential points.
- **Assumptions of Least Squares Regression:** Analyzing the fit also involves checking if the assumptions underlying ordinary least squares (OLS) regression are reasonably met. These assumptions include linearity, independence of errors, homoscedasticity, and normality of errors. The diagnostic tools mentioned above help in assessing these assumptions.

- **Refining the Model:** If the analysis of the fit reveals issues (e.g., non-linearity, heteroscedasticity, influential outliers), the model might need to be refined. This could involve transforming variables, adding or removing predictors, or using different modeling techniques.

In summary, analyzing the fit of a model in R involves using functions like `summary()`, extracting specific model information with functions like `coef()`, `residuals()`, and `anova()`, and examining diagnostic plots generated by `plot()` to assess the model's adequacy and the validity of its underlying assumptions. This process helps in understanding the model's performance and identifying potential areas for improvement.

Refining the model

Refining a model in R is the process of making adjustments to an initially built model to improve its fit, predictive power, or adherence to the underlying assumptions. This is often done after analyzing the initial fit of the model and identifying potential areas for improvement.

Here's a detailed breakdown of refining a model based on the sources:

- **When Refinement is Needed:** The need to refine a model arises when the analysis of the fit reveals issues. These issues could include:

    - Violation of the assumptions of least squares regression (e.g., non-linearity, heteroscedasticity, non-normality of errors).
    - Presence of unusual observations (outliers or high-leverage points) that disproportionately influence the model. The syllabus mentions "Unusual observation and corrective measures" as a step in Module 5.
    - Inclusion of insignificant predictor variables, leading to a more complex model than necessary.
    - A model that doesn't adequately explain the variance in the response variable.

- **Using the `update()` function:** One common and efficient way to refine a model in R is by using the **`update()` function**. This function allows you to refit a model by making changes to the original formula or the data used.
    - The `update()` function takes the **existing model object** as its first argument.
    - You can then specify the changes you want to make, such as adding or subtracting terms from the formula.
    - It can also be used to update the data frame used for fitting.
    - Using `update()` can **save typing** in interactive R sessions and can be more **computationally efficient** for large datasets compared to refitting the entire model from scratch.

```
# Assuming 'initial.model' is your fitted model object

refined.model <- update(initial.model, . ~ . + new_predictor) # Add a predictor

refined.model_no_intercept <- update(initial.model, . ~ . - 1) # Remove the intercept

refined.model_different_data <- update(initial.model, data = new_data_frame) # Use a new data frame
```

- **Addressing Unusual Observations:** The syllabus specifically mentions "corrective measures of unusual observations in regression modelling". While the excerpts don't detail these measures, "R in a Nutshell" mentions that diagnostic plots (obtained using `plot()` on a fitted `lm` object) can help identify influential observations using Cook's distance and residuals vs leverage plots. Corrective measures might involve:
    - Investigating the reasons for the unusual observations (e.g., data entry errors).
    - Removing the observations if they are deemed erroneous and not representative of the population.
    - Using robust regression techniques (discussed in Chapter 20) that are less sensitive to outliers. The `rlm()` function in the `MASS` package is mentioned for robust linear model fitting.
- **Refining the Model Formula:** Based on the analysis of the fit (e.g., coefficient significance from `summary()` or model comparison using AIC), you might decide to change the model formula. This could involve:
    - **Adding predictor variables** that might explain more of the variance in the response.
    - **Removing predictor variables** that are not statistically significant or do not contribute meaningfully to the model.
    - **Transforming variables** (e.g., taking logarithms or creating polynomial terms) to address non-linear relationships. While the syllabus asks "polynomial in linear

regression?", the excerpt doesn't provide details on how to implement this, but it's implied that predictors in `lm()` can be nonlinear functions of variables.
    - Considering **interaction terms** between predictor variables if their combined effect is different from their individual effects.
- **Model Selection Techniques:** "R in a Nutshell" discusses several techniques for refining a model by selecting a subset of the most important variables:
    - **Stepwise Variable Selection:** The `step()` function can automatically add or remove variables from a model based on a criterion like AIC to find a model with a good balance of fit and complexity. The `stepAIC()` function in the `MASS` library is also mentioned as an alternative.
    - **Shrinkage Methods:** Techniques like **ridge regression** (implemented by `select()` in the `MASS` package) and **lasso** (available through the `lars()` function and `glmnet()` function) can shrink the coefficients of less important variables towards zero, effectively performing variable selection and potentially improving prediction accuracy.
- **Comparison of Models:** After refining a model, it's often necessary to **compare it with the original model or other refined versions**. This can be done using various criteria:
    - **Adjusted R-squared:** A higher adjusted R-squared indicates a better fit, considering the number of predictors in the model.
    - **AIC (Akaike Information Criterion):** Lower AIC values generally indicate a better model in terms of prediction accuracy relative to its complexity. The `extractAIC()` function can be used to compute AIC.
    - **F-tests and p-values:** Comparing nested models using the `anova()` function can assess whether adding or removing a set of predictors significantly improves or worsens the model fit.
    - **Diagnostic plots:** Comparing the residual plots of different models can help assess which model better meets the regression assumptions.

In summary, refining a model in R is an iterative process that involves analyzing the initial model fit, identifying shortcomings, and applying various techniques like updating the formula or data, addressing unusual observations, using model selection methods, and comparing different model specifications to arrive at a more suitable and reliable model.

Regression- types **[A revision of all the other topics included]**

#Module 5 of the "AIT 362 Programming in R Syllabus" focuses on **Regression Models**. The syllabus outlines several key aspects of regression, including:

- **Building linear models** - This involves creating models where the relationship between the response variable and the predictors is assumed to be linear. The "R in a Nutshell" book explains that a linear regression assumes a response variable *y* is a linear function of predictor variables *x1, x2, ..., xn*, with coefficients and an error term. The `lm()` function in R is used to fit linear models.
- **Predict values using models** - Once a regression model is built, it can be used to predict the response variable for new data based on the predictor variables. The `predict()` function in R can be used for this purpose, and it can also return confidence intervals for predictions.
- **Analyzing the fit** - After fitting a model, it's important to assess how well it explains the data. The `summary()` function in R provides detailed information about the model fit, including coefficients, standard errors, t-values, p-values, and measures like R-squared. Diagnostic plots can be generated using the `plot()` function to assess residuals and other aspects of the fit.
- **Refining the model** - Based on the analysis of the fit, the model may need to be adjusted or refined to improve its performance. This can involve transforming variables, adding or removing predictors, or considering different types of regression models .#

- **Regression- types** - The syllabus explicitly mentions exploring different types of regression. The "R in a Nutshell" book provides details on various regression types:
  - **Linear Regression:** As discussed earlier, this is the fundamental type assuming a linear relationship between variables.
  - **Generalized Linear Models (GLMs):** This framework extends linear regression to handle response variables with non-normal error distributions. The syllabus specifically mentions two types of GLMs:
    - **Logistic Regression:** Used for binary response variables (e.g., success/failure, yes/no) by modeling the probability of the outcome using a logistic function. In R, the `glm()` function with the `binomial` family is used for logistic regression.
    - **Poisson Regression:** Used for modeling count data (non-negative integers) by assuming a Poisson distribution for the response variable. The `glm()` function with the `poisson` family in R is used for Poisson regression .
  - The "R in a Nutshell" book also covers other GLM families like Gaussian, Gamma, and inverse Gaussian, which can be used for different types of continuous response variables.
  - **Robust and Resistant Regression:** These methods are useful when the data contains outliers or violates the assumption of homoscedasticity (constant variance of errors). Functions like `rlm` in the `MASS` package and `lqs` are available in R for these types of regression.
  - **Ridge Regression:** A technique used to handle multicollinearity (high correlation between predictor variables) by adding a penalty term to the least squares estimation. The `lm.ridge` function in the `MASS` package can be used.

- **Lasso and Least Angle Regression:** These are other shrinkage methods that can perform variable selection by forcing some coefficients to be exactly zero. The `lars` package provides functions for these methods.
- **Principal Components Regression and Partial Least Squares Regression:** These techniques are also used when dealing with multicollinear predictors by first transforming the predictors into a smaller set of uncorrelated components and then performing regression. The `pcr` and `plsr` functions in the `pls` package can be used.
- **Nonlinear Least Squares:** Used to model relationships that are not linear in the parameters. The `nls()` function in R is used for fitting nonlinear models .

#

- **Unusual observation and corrective measures** - Regression analysis can be affected by unusual observations such as outliers or high-leverage points. The "R in a Nutshell" book discusses methods for identifying these observations and corrective measures that can be taken, potentially involving robust regression techniques or removing influential points . The `influence.measures()` function can provide statistics to identify influential observations.
- **Comparison of models** - When multiple regression models are built, it's often necessary to compare them to choose the best one. Criteria like AIC (Akaike Information Criterion), residual standard error, and R-squared can be used for comparison. The `anova()` function can also be used to compare nested linear models.
- **Generalized linear models - Logistic Regression, Poisson Regression** - As mentioned earlier, these are specific types of regression models within the broader class of GLMs.
- **Nonlinear least squares** - This technique is used to fit models where the relationship between the dependent and independent variables is nonlinear.

In summary, Module 5 introduces fundamental concepts of regression modeling in R and delves into various types of regression techniques suitable for different data characteristics and research questions. The "R in a Nutshell" book provides a more comprehensive treatment of these topics, offering theoretical background and practical guidance on implementing these regression models using R functions and packages.#

Unusual observation and corrective measures

The inclusion of this topic indicates its importance in understanding and building effective regression models in R.

"R in a Nutshell," in its Chapter 20 on Regression Models, provides practical details and R functions relevant to identifying and addressing unusual observations.

Identifying Unusual Observations:

"R in a Nutshell" explains that after fitting a linear model (using functions like `lm()`), it's crucial to analyze the fit. A key part of this analysis involves identifying unusual observations that might disproportionately influence the model. These unusual observations can be broadly categorized as:

- **Outliers:** Data points where the observed response value is far from the value predicted by the model, resulting in large residuals (the difference between observed and predicted values) .
- **High-leverage points:** Observations with unusual or extreme values for the predictor variables. These points have the potential to exert a strong influence on the estimated regression coefficients.
- **Influential points:** Observations that, if removed from the dataset, would significantly change the model's coefficients, fit statistics, or predictions. These points can be a combination of outliers and high-leverage points.

"R in a Nutshell" highlights that **diagnostic plots**, generated using the `plot()` function on a fitted `lm` object, are essential tools for identifying these unusual observations. Some of the key diagnostic plots and what they reveal are:

- **Residuals vs Fitted:** This plot can help identify outliers (points far from the zero line) and non-linear patterns, as well as non-constant variance (heteroscedasticity).
- **Normal Q-Q:** This plot assesses the normality of residuals. Deviations from the straight diagonal line may indicate outliers or non-normal errors.
- **Scale-Location (Spread vs Fitted):** This plot helps in detecting non-constant variance of the residuals. A non-horizontal pattern suggests heteroscedasticity.
- **Cook's Distance Plot:** This plot visualizes Cook's distance for each observation, which measures the overall influence of a data point on the model. Points with high Cook's distance (often above a threshold like 0.5 or 1) are considered influential.
- **Residuals vs Leverage Plot:** This plot helps identify both outliers (large absolute residuals) and high-leverage points (high leverage values on the x-axis). Influential points often appear in the upper right or upper left corners of this plot.

The syllabus also mentions "Regression- types", and different types of regression might be more or less sensitive to unusual observations. For instance, ordinary least squares (OLS) linear regression, often the starting point ("Building linear models - model fitting"), can be quite sensitive to outliers.

Corrective Measures for Unusual Observations:

When unusual observations are identified, "R in a Nutshell" and the syllabus's mention of "corrective measures" suggest several approaches:

1. **Investigate the Data:** The first step should always be to understand why these observations are unusual. It could be due to data entry errors, measurement mistakes, or genuinely unusual but valid data points. If an error is found, the data should be corrected.
2. **Remove Erroneous Observations:** If the unusual observations are due to clear errors and are not representative of the population, they might be removed from the dataset before refitting the model. However, this should be done with caution and justification.
3. **Transform Variables:** In some cases, transforming the response variable or predictor variables (e.g., using a logarithmic or square root transformation) can reduce the impact of outliers and address issues like non-linearity or heteroscedasticity. The syllabus asks "polynomial in linear regression?", suggesting that non-linear relationships can be modeled in a linear regression framework using polynomial terms, which might help in certain cases of unusual observations arising from non-linear patterns.
4. **Use Robust Regression Techniques:** "R in a Nutshell" specifically discusses **robust and resistant regression** methods as alternatives less sensitive to outliers. The `rlm()` function in the `MASS` package is mentioned for fitting linear models using MM-estimation, which provides more robust coefficient estimates in the presence of outliers. Another robust regression function mentioned is `lqs()` also in the `MASS` package.
5. **Winsorizing or Trimming Data:** Winsorizing involves replacing extreme values with values closer to the center of the distribution, while trimming involves removing a certain percentage of the most extreme values. These techniques can reduce the influence of outliers but should be used thoughtfully.
6. **Consider Different Model Types:** If linear regression is consistently affected by unusual observations, exploring other types of regression models (as indicated by "Regression- types" in the syllabus), such as non-linear regression or generalized linear models (GLMs) like Poisson or Logistic Regression (also mentioned in Module 5), might be appropriate depending on the nature of the data and the research question.

The syllabus also includes "Comparison of models", which becomes relevant when considering corrective measures. After applying a corrective measure (e.g., removing an outlier or using robust regression), the refined model should be compared to the original model to assess whether the changes have led to an improvement in the model's fit and predictive ability. This comparison can involve examining summary statistics (like Adjusted R-squared), conducting statistical tests, and comparing diagnostic plots.

In summary, within Module 5 of the AIT 362 Programming in R course, understanding and addressing unusual observations in regression models is a critical component. This involves using R tools like diagnostic plots to identify outliers, high-leverage points, and influential observations, and then applying appropriate corrective measures such as data investigation, removal (with caution), variable transformations, or employing robust regression techniques as discussed in "R in a Nutshell". The process often involves comparing different model specifications to determine the most suitable and reliable model.

Comparison of models

The need for comparing models arises after you have built and potentially refined one or more regression models. As Module 5 also covers "Regression- types" and "Refining the model", you might have different models to choose from, whether they are variations of linear models or entirely different types of regression models. The goal of model comparison is to determine which model best explains the data, has better predictive power, and is more suitable for the research question at hand.

Here are some aspects of comparing regression models:

- **Why Compare Models?** The primary reason to compare models is to select the most appropriate model for your data and objectives. This involves considering factors like the model's fit to the data ("Analyzing the fit"), its predictive accuracy ("Predict values using models"), and its complexity. You might also want to compare a simpler model with a more complex one to see if the added complexity provides a significant improvement in fit or prediction.

- **Methods for Comparing Models:**

    - **Statistical Tests:** When comparing nested linear models (where one model is a special case of another), you can use the **`anova()` function** to perform an analysis of variance test. According to "R in a Nutshell," for linear models, the `anova.lmlist` method is used, and by default, it includes F-test statistics to compare the mean square for each row to the residual mean square. You can also specify other test statistics like chi-squared tests. The syllabus mentions "Comparison of models" after "Building linear models - model fitting" and "Refining the model", suggesting that `anova()` would be a key tool when deciding if refining a model (e.g., adding more terms) significantly improves it.
    - **Fit Statistics:** Various statistics are used to assess how well a model fits the data. **Adjusted R-squared** is often used to compare models with different numbers of predictors, as it penalizes the inclusion of unnecessary variables. The output of a fitted linear model in R (using `lm()`) provides R-squared and adjusted R-squared values. Other fit statistics, such as **Akaike Information Criterion (AIC)** and **Bayesian Information Criterion (BIC)**, can also be used to compare non-nested models, with lower values generally indicating a better model considering the trade-off between fit and complexity. The `step()` function in R, used for stepwise variable selection (a model refinement technique), uses AIC as a criterion for adding or removing variables. Generalized linear models fitted with `glm()` also provide AIC values in their output.
    - **Residual Analysis and Diagnostic Plots:** Comparing the **residuals** of different models is crucial. Models with randomly distributed residuals (no discernible patterns) and consistent variance are generally preferred. The `plot()` function applied to a fitted `lm`

object generates diagnostic plots like "Residuals vs Fitted" and "Normal Q-Q" plots, which help in assessing the model's assumptions and identifying potential issues. Comparing these plots across different models can inform the choice of the best model.

  - **Predictive Performance:** If the primary goal is prediction ("Predict values using models"), comparing the **predictive accuracy** of different models on unseen data (testing data) is essential. This often involves splitting the data into training and testing sets, fitting models on the training data, and then evaluating their performance on the testing data using metrics like **Root Mean Squared Error (RMSE)**. While the syllabus doesn't explicitly detail these metrics, the emphasis on prediction in Module 5 implies their relevance in model comparison.
  - **Comparison of Different Regression Types:** Module 5 covers "Regression- types" and "Generalized linear models - Logistic Regression, Poisson". You might need to compare a linear regression model with a generalized linear model if the assumptions of linearity or normality are violated. The choice would depend on which model better addresses the nature of the response variable and the underlying relationships in the data.
- **R Functions for Model Comparison:**
  - **`anova()`:** As mentioned, for comparing nested linear models.
  - **`summary()`:** Provides key fit statistics like R-squared, adjusted R-squared, and coefficients with their standard errors and p-values, allowing for assessment of individual predictor significance and overall model fit.
  - **`AIC()` and `BIC()`:** Functions (often part of the `stats` package) to calculate these information criteria for model comparison (though not explicitly detailed in the provided excerpts, they are standard tools in R for this purpose).
  - **`predict()`:** Used to generate predictions from different models on the same dataset or a new dataset, allowing for a direct comparison of their predictive capabilities.
  - **`plot()`:** For generating diagnostic plots to visually assess the adequacy of different models.
  - Functions from packages like `caret` (not in the excerpts) provide more advanced tools for model comparison, including cross-validation for robust assessment of predictive performance.

In summary, "Comparison of models" in Module 5 is a crucial step in the regression modeling process. It involves using statistical tests like `anova()` for nested models, comparing fit statistics (R-squared, adjusted R-squared, AIC, BIC), analyzing residuals and diagnostic plots generated by `plot()`, and evaluating predictive performance using functions like `predict()`. This comprehensive comparison enables you to select the most suitable regression model from the various types and refined versions you might have explored in this module.

Generalized linear models - Logistic Regression, Poisson Regression

Module 5 of the syllabus covers various aspects of regression models, and it specifically includes "Generalized linear models - Logistic Regression, Poisson" as a key topic. This topic is allocated one lecture hour, suggesting it's a significant part of understanding regression beyond standard linear models.

Generalized Linear Models (GLMs):

Generalized linear modeling, as mentioned in "R in a Nutshell," is a framework that allows for the fitting of many common types of models using a single approach. It was developed to handle situations where the assumptions of ordinary linear regression (like the response variable being continuous and normally distributed) are not met.

Here are the key components of a GLM:

- **A random component:** This specifies the probability distribution of the response variable (y). Unlike linear models that assume a normal distribution for the errors, GLMs allow for other distributions from the exponential family, such as binomial, Poisson, gamma, etc..
- **A systematic component (linear predictor):** This is a linear combination of the predictor variables ($x_1, x_2, ..., x_n$) and their coefficients ($c_1, c_2, ..., c_n$), along with an intercept (c_0). It takes the form: $\eta = c_0 + c_1x_1 + c_2x_2 + ... + c_nx_n$. The relationship between the response and predictor variables doesn't have to be linear, but the relationship between the predictor variables and the linear predictor must be linear.
- **A link function:** This is a smooth, invertible function (denoted as *l*) that defines the relationship between the expected value (mean, μ) of the response variable and the linear predictor (η): $\eta = l(\mu)$. Equivalently, $\mu = m(\eta)$, where *m* is the inverse of the link function. The link function connects the random component to the systematic component.

The `glm()` function in R is used to fit generalized linear models. You specify the formula, the data, and the `family` argument, which indicates both the probability distribution of the response and the link function to be used.

Logistic Regression:

Logistic regression is a type of GLM specifically used when the response variable is **categorical with two outcomes** (binary). The goal is often to model the probability of one of these outcomes occurring.

Key aspects of logistic regression:

- **Response Variable:** Binary (e.g., success/failure, yes/no, good/bad).
- **Probability Modeling:** It models the probability of a certain outcome (say, outcome A) as a function of the predictor variables. Instead of directly modeling the probability linearly (which could lead to values outside the 0 to 1 range), it uses a link function that constrains the predicted values to be within this range.

- **Link Function:** The most common link function for logistic regression is the **logit function**, which is the logarithm of the odds of the outcome occurring: $logit(p) = log(\frac{p}{1-p}) = \eta$, where *p* is the probability of the outcome. The inverse of the logit function is the **logistic function**: $p = \frac{1}{1 + e^{-\eta}}$.
- **R Implementation:** In R, you fit a logistic regression model using the `glm()` function with the `family = "binomial"` argument. By default, the `"binomial"` family uses the `"logit"` link function, but you can specify other links like `"probit"` or `"cloglog"` if needed.
- **Interpretation:** The coefficients in a logistic regression model represent the change in the log-odds of the outcome for a one-unit change in the predictor variable. Exponentiating these coefficients gives the odds ratios.
- **Output:** The `summary()` of a `glm()` object fitted with the binomial family provides information such as coefficients, standard errors, z-values, and p-values. It also includes deviance residuals, AIC, null deviance, and residual deviance.

Poisson Regression:

Poisson regression is another type of GLM used to model **count data**. Count data are non-negative integers, often representing the number of times an event occurs in a fixed interval of time or space.

Key aspects of Poisson regression:

- **Response Variable:** Count data (non-negative integers).
- **Distribution:** The Poisson distribution is used to model the response variable. The Poisson distribution assumes that the mean and variance of the count data are equal.
- **Link Function:** The most common link function for Poisson regression is the **log link**: $\eta = log(\mu)$, where μ is the expected count. The inverse of the log link is the exponential function: $\mu = e^{\eta}$.
- **R Implementation:** You fit a Poisson regression model in R using the `glm()` function with the `family = "poisson"` argument. By default, the `"poisson"` family uses the `"log"` link function. You can specify other link functions if theoretically justified.
- **Interpretation:** The coefficients in a Poisson regression model, when the log link is used, represent the change in the log of the expected count for a one-unit change in the predictor variable. Exponentiating these coefficients gives the rate ratios (incidence rate ratios).
- **Output:** Similar to logistic regression, the `summary()` of a `glm()` object with the Poisson family provides coefficient estimates, standard errors, z-values, p-values, AIC, null deviance, and residual deviance.

Comparison and Context in Module 5:

The inclusion of "Generalized linear models - Logistic Regression, Poisson" in Module 5, after topics like building and refining linear models, suggests a progression in understanding regression techniques. You would likely learn about the limitations of standard linear regression when dealing with non-continuous or non-normally distributed response variables, leading to the introduction of GLMs as a more flexible framework.

The syllabus also mentions "Regression- types", and logistic and Poisson regression would fall under this broader categorization of regression models. Understanding these different types of regression models allows for a more nuanced approach to analyzing various types of data encountered in statistical analysis.

The `glm()` function serves as a central tool for implementing these models in R, highlighting the modularity and flexibility of R for statistical analysis. The output and interpretation of these models differ from linear regression (e.g., focusing on probabilities or expected counts rather than direct prediction of a continuous variable), which would be a key aspect of learning this topic in Module 5. The mention of "Comparison of models" in Module 5 also implies that you would learn how to compare the fit and appropriateness of GLMs versus linear models, or even different GLMs, for a given dataset.

Nonlinear least squares

According to "R in a Nutshell," **Nonlinear Least Squares** is a method used to **fit nonlinear models**. This technique is applicable when you know the general form of a model, even if that model exhibits extreme nonlinearity. The goal of nonlinear least squares is to **minimize the least squares error** when fitting such models to data.

Here are key aspects of nonlinear least squares as described in the sources:

- **Use Case:** It is employed when the relationship between the response and predictor variables cannot be adequately represented by a linear model. Instead, a specific, potentially complex, nonlinear functional form is assumed.
- **R Function:** In R, the `nls()` function is used to determine the nonlinear least squares estimates of the parameters of a nonlinear model.

Syntax of `nls()`: The `nls()` function in R has the following general syntax:

```
nls(formula, data, subset, weights, na.action, start,

  control = nls.control(...), algorithm, trace, ...)
```

- **Key Arguments of `nls()`:**
    - `formula`: This argument specifies the nonlinear model to be fitted. It's a symbolic description of the relationship between the response variable and the predictor variables, including the parameters to be estimated.
    - `data`: This is a data frame, list, or environment containing the variables used in the formula.
    - `start`: This is a crucial argument that provides **starting values for the parameters** to be estimated. The success and convergence of the nonlinear least squares algorithm often depend on providing reasonable starting values.

    - **`algorithm`**: This argument specifies the algorithm to be used for fitting the model. Two common options are:
        - `"plinear"`: The Golub-Pereyra algorithm for partially linear least squares models.
        - `"port"`: The 'nl2sol' algorithm from the PORT library.
    - **`trace`**: A logical value indicating whether to print the progress of the algorithm while `nls` is running.
    - **`subset`**: An optional vector specifying the rows to include in the fit.
    - **`weights`**: An optional vector specifying weights for the observations.
    - **`control`**: A list of control parameters for the nonlinear least squares algorithm, which can be set using the `nls.control()` function.
- **Contrast with Linear Models:** Unlike linear models fitted with `lm()`, where a direct mathematical solution for the coefficients exists, nonlinear least squares typically involve iterative optimization algorithms. These algorithms start with initial guesses for the parameter values and iteratively refine them to minimize the sum of squared differences between the observed and predicted values.
- **Complexity:** Fitting nonlinear models can be more challenging than fitting linear models due to potential issues with convergence to a global minimum, sensitivity to starting values, and the need to specify the model form correctly.

In summary, within Module 5 on Regression Models, you would learn that nonlinear least squares is a powerful technique in R, implemented through the `nls()` function, to model relationships between variables that are not linear. This method requires specifying a formula that describes the nonlinear relationship and providing starting estimates for the model parameters. The module would likely cover the importance of choosing appropriate starting values and understanding the output of the `nls()` function to assess the quality of the model fit.
