Nachhol-Hausarbeit

Organisatorisches

- Diese Hausarbeit besteht aus 5 Aufgaben mit einer Gesamtpunktzahl von 150 Punkten.
- Durch die Abgabe dieser Hausarbeit können maximal 85 % der Gesamtpunktzahl für den Kurs „Programmieren mit statistischer Software“ im Sommersemester 2017 erreicht werden.
- Die individuell erreichte Punktzahl für den Kurs „Programmieren mit statistischer Software“ setzt sich aus der erreichten Punktzahl durch abgegebene Übungsaufgaben (maximal 15 %) und der erreichten Punktzahl durch die Bearbeitung und Abgabe dieser Hausarbeit (maximal 85 %) zusammen.
- Eine verbindliche Anmeldung zur Veranstaltung (d.h. Scheinerwerb) erfolgt mit Abgabe dieser Hausarbeit.
- **Letztmöglicher Abgabetermin: Freitag, 22. Dezember 2017 (23:55 Uhr).**

Hinweise zur Bearbeitung und Abgabe

- Bitte geben Sie die Hausarbeit über Moodle ab. Eine Abgabe ist nur möglich, falls Sie sich für den Kurs eingeschrieben haben. Bitte beachten Sie, dass die Selbsteinschreibung in den Kurs nur noch bis zum 31.07.2017 möglich ist. Danach muss eine manuelle Einschreibung über die Kursleitung erfolgen. Schreiben Sie dazu bitte eine formlose E-Mail an eva.endres@stat.uni-muenchen.de.
- Verwenden Sie für Ihre Lösung ausschließlich die Vorlagendatei „ProgStat17_Nachname_Vorname.R“. Benennen Sie die Datei entsprechend mit Ihrem Nach- und Vornamen.
- Tragen Sie in die R-Syntax-Datei „ProgStat17_Nachname_Vorname.R“ an ausgewiesener Stelle alle prüfungsrelevanten Informationen (vollständiger Name, Matrikelnummer, Studiengang, Prüfungsordnung) ein. Geben Sie auch an, ob Sie mit einer Veröffentlichung Ihrer Note auf Moodle einverstanden sind oder nicht.
- Machen Sie unbedingt deutlich, welcher Programm-Code zu welcher (Teil-) Aufgabe gehört.
- Achten Sie darauf, dass Ihr Programm-Code nachvollziehbar und ordentlich dokumentiert und kommentiert ist.
- Alle Teilschritte, die aufgrund eines fehlerhaften Programm-Codes eine nicht beabsichtigte Fehlermeldung erzeugen, werden mit 0 Punkten bewertet.

Aufgabe 1 (Datenanalyse) [20 Punkte]

a) Laden Sie den Datensatz `mtcars` aus dem `datasets`-Paket. Rufen Sie die zugehörige Dokumentation auf und machen Sie sich mit den Daten vertraut.

b) Speichern Sie den Datensatz in einem Objekt namens `mycars` ab.

c) Verschaffen Sie sich einen Überblick über die Daten.

d) Kodieren Sie alle kategorialen Variablen explizit als solche und weisen Sie ihnen (aussagekräftige) Wertelabel zu. Geben Sie für jede kategoriale Variable eine relative Häufigkeitstabelle aus.

e) Speichern Sie die aktuelle Version dieses Datensatzes in Ihr aktuelles Arbeitsverzeichnis.

f) Extrahieren Sie die Herstellernamen (z. B. „Mazda“, „Merc“, „Toyota“ usw.) der in dem Datensatz enthaltenen Fahrzeuge mithilfe eines *regulären Ausdrucks*. Speichern Sie die Herstellerbezeichnungen in einer eigenen Variable `hersteller` in dem Datensatz ab.

g) Verfahren Sie wie in Teilaufgabe f) uns speichern Sie alle Modellnamen (z.B. „RX4“, „RX4 Wag“, „710“, usw.) in einer eigenen Variable `modell` in dem Datensatz ab. Verwenden Sie auch hierzu einen *regulären Ausdruck*.

h) Geben Sie die absoluten Häufigkeiten der `hersteller` in der Konsole aus.

i) Berechnen Sie alle Quartile der in dem Datensatz vorhandenen numerischen Variablen. Verwenden Sie dazu eine geeignete Funktion der `apply`-Familie. Das Ergebnis soll in Form einer `matrix` ausgegeben werden.

j) Berechnen Sie die mediane Zylinderzahl für alle Beobachtungen mit manuellem Getriebe und einer Mindestreichweite von 20 Meilen pro Galleone.

Aufgabe 2 (Textverarbeitung) [40 Punkte]

Die Wissenschaft der Kryptologie beschäftigt sich mit der Ver- und Entschlüsselung von Informationen. Die Caesar-Verschlüsselung ist eine Methode, die zu den einfachen Substitutionsverfahren zählt[1]. Dabei wird ein Alphabet von Klartextbuchstaben abgebildet in ein Alphabet von Geheimtextbuchstaben. Diese Abbildung erfolgt zyklisch durch eine Verschiebung um k Positionen nach links. Betrachtet man zum Beispiel das Alphabet der Kleinbuchstaben, dann ergibt sich mit $k = 3$ folgende (zyklische) Verschiebung:

Klartext	a	b	c	d	...	x	y	z
Geheimtext	d	e	f	g	...	a	b	c

Aus dem Wort „klartext“ wird in Geheimschrift dann das Wort „noduwhaw“.

a) Schreiben Sie eine Funktion `encrypt()`, die einen Klartext mit der Caesar-Verschlüsselung chiffriert. Die Grundstruktur der Funktion soll folgendermaßen aussehen:

```
encrypt <- function(text, verschiebung){
...
...
...
}
```

Das Argument `text` dient zur Übergabe des Klartextes, das Argument `verschiebung` gibt die Anzahl der Positionen an, um die ein Alphabet rotiert werden soll.
Rückgabe der Funktion ist eine Liste mit folgenden Elementen:

- Den originalen Klartext,
- den verschlüsselten Geheimtext,
- die Verschiebung, die zur Verschlüsselung verwendet wurde.

Die Funktion soll außerdem folgende Spezifikationen erfüllen:

- Entgegen obigem Beispiel sind die zu verschüsselnden Zeichen nicht nur Kleinbuchstaben, sondern (**in dieser Reihenfolge!**) Großbuchstaben, Kleinbuchstaben, Leerzeichen und die Satzzeichen „Punkt“, „Komma“, „Semikolon“, „Ausrufezeichen“ und „Fragezeichen“.
- Die Funktion soll keine Umlaute berücksichtigen.
- Die Elemente des Outputs sollen sinnvoll bezeichnet werden.
- Der Output der Funktion soll von der Objektklasse `encrypt` sein.

[1] siehe z. B. F.L. Bauer: Entzifferte Geheimnisse: Methoden und Maximen der Kryptologie, 2. erweiterte Auflage, Springer-Verlag, Berlin, 1997.

b) Schreiben Sie eine Funktion `decrypt()`, die einen mit der Caesar-Verschlüsselung chiffrierten Text wieder in Klartext umwandelt. Die Grundstruktur der Funktion soll folgendermaßen aussehen:

```
decrypt <- function(text, verschiebung){
...
...
...
}
```

Das Argument `text` dient zur Übergabe des Geheimtextes, das Argument `verschiebung` gibt die Anzahl der Positionen an, um die das Alphabet rotiert werden muss.
Rückgabe der Funktion ist eine Liste mit folgenden Elementen:

- Den originalen Geheimtext,
- den entschlüsselten Klartext,
- die Verschiebung, die zur Verschlüsselung verwendet wurde.

Die Funktion soll außerdem folgende Spezifikationen erfüllen:

- Entgegen obigem Beispiel sind die zu entschüsselnden Zeichen nicht nur Kleinbuchstaben, sondern (**in dieser Reihenfolge!**) Großbuchstaben, Kleinbuchstaben, Leerzeichen und die Satzzeichen „Punkt“, „Komma“, „Semikolon“, „Ausrufezeichen“ und „Fragezeichen“.
- Die Funktion soll keine Umlaute berücksichtigen.
- Die Elemente des Outputs sollen sinnvoll bezeichnet werden.
- Der Output der Funktion soll von der Objektklasse `decrypt` sein.

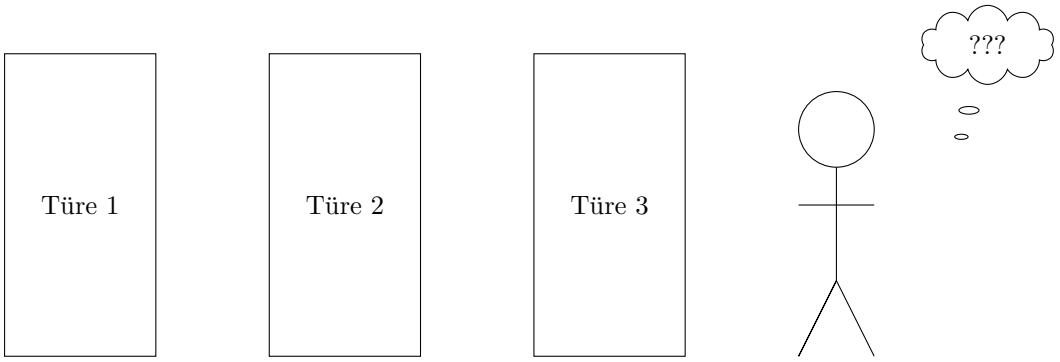
c) Ergänzen Sie den Output der beiden Funktionen `encrypt` und `decrypt` an geeigneter Stelle um die Objektklasse `caesar` und testen Sie, ob Ihr Code korrekt arbeitet.

d) Definieren Sie eine `print`-Methode für die Objektklasse `caesar`, die den Klartext, den Geheimtext und die Verschiebung übersichtlich in der Konsole ausgibt.

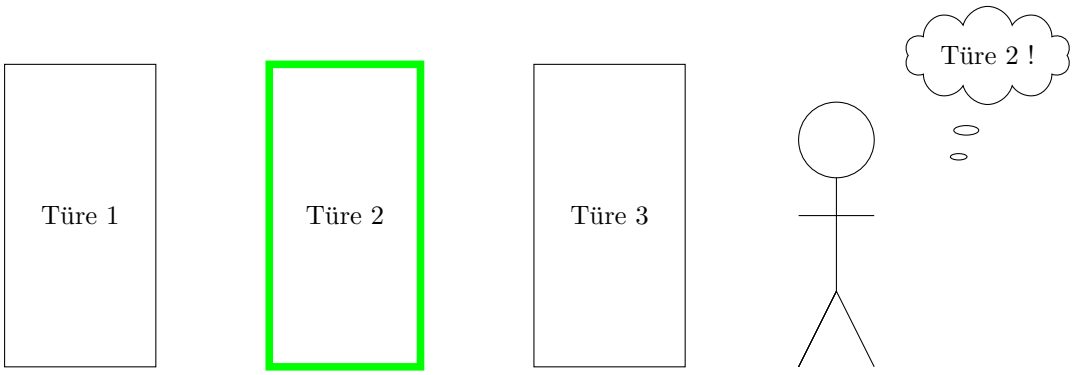
e) Speichern Sie folgenden Beispieltext als einziges Element in einem `character`-Vektor ab:
`lt,,JxrwJx,Jtx,tAJeADtuD,vJpqBrwAtxqtLJup  tJxrwJsDArw`
Dechiffrieren Sie diesen Text bei einer Verschiebung um 15 Zeichen und geben Sie das Ergebnis in der Konsole mithilfe der `print`-Funktion aus.
Hinweis: Zwischen den Zeichen „p“ und „t“ (im Beispieltext) befinden sich zwei Leerzeichen.

Aufgabe 3 (Simulation I) [30 Punkte]

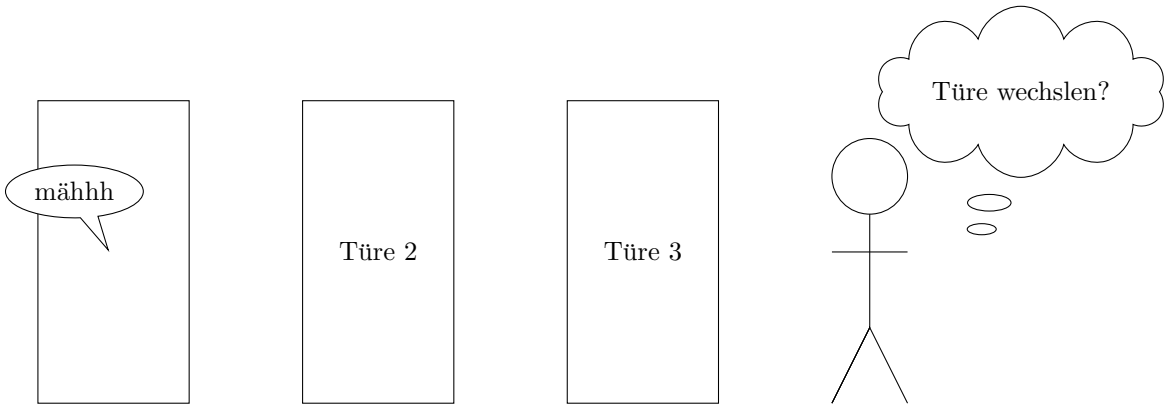
Das „Ziegenproblem“:
Stellen Sie sich vor, Sie nehmen an einer Spielshow teil. Der Showmaster zeigt Ihnen drei Türen. Hinter einer der Türen verbirgt sich der Hauptpreis, hinter den beiden anderen wartet jeweils eine Ziege auf Sie. Sie sollen sich für eine der drei Türen entscheiden. Ihr Ziel ist es, den Hauptpreis zu erhalten.



Angenommen, Sie treffen eine Entscheidung und wählen Türe 2.



Nachdem Sie Ihre Entscheidung dem Showmaster mitgeteilt haben, öffnet der Showmaster Türe 1. Eine Ziege kommt zum Vorschein. Der Showmaster gibt Ihnen nun die Möglichkeit Ihre Entscheidung nochmal zu überdenken. Was machen Sie? Bleiben Sie bei der ursprüglich gewählten Türe 2 oder ändern Sie Ihre Meinung und wechseln zu Türe 3?



a) Schreiben Sie eine Funktion `simulation.ziegenproblem()`, die obige Spielsituation `n` mal erzeugt. Die Funktion soll folgende Spezifikationen erfüllen:

- Die Funktion soll folgende Argumente enthalten:
  - die Anzahl `n` an zu erstellenden Spielsituationen,
  - die Strategie `strategie` für diese Spielsituationen.
- Eine Spielsituation entspricht der obigen Beschreibung. Es wird zwischen drei Türen gewählt. Der Showmaster öffnet eine Türe, hinter der eine Ziege steht. Nachdem diese Türe geöffnet wurde, gibt es zwei mögliche Strategien:
  - Strategie 1, `behalten`: Die ursprüngliche Entscheidung für eine Türe wird beibehalten.
  - Strategie 2, `wechseln`: Die ursprüngliche Entscheidung für eine Türe wird verworfen und die andere noch verschlossene Türe gewählt.
- Zum Verhalten des Showmasters: Der Showmaster kennt die Platzierungen der Ziegen und des Hauptpreises hinter den Türen. Er wird nie die Türe des Hauptpreises öffnen, sondern immer eine Türe, hinter der sich eine Ziege verbirgt.
- Die Funktion soll den Anteil der Spielsituationen ausgeben, die zum Hauptpreis geführt haben.

b) Verwenden Sie die in Teilaufgabe a) erstellte Funktion und simulieren Sie sich jeweils 200 Spielsituationen für die beiden möglichen Strategien. Speichern Sie die Ergebnisse in den beiden Objekten `sim.behalten` bzw. `sim.wechseln` und vergleichen Sie sie. Achten Sie darauf, dass ihre Ergebnisse reproduzierbar sind!

c) Wiederholen Sie beide Simulationen aus Teilaufgabe b) 100 mal und stellen Sie die Ergebnisse auf geeignete Weise grafisch dar und interpretieren Sie sie.

Aufgabe 4 (Newton-Raphson) [30 Punkte]

Das Newton-Raphson-Verfahren ist ein Näherungsverfahren zur Bestimmung einer Nullstelle einer konvexen, stetig differenzierbaren Funktion $f : \mathcal{I} \to \mathbb{R}$. Die Iterationsvorschrift lautet

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

f bezeichnet die Funktion, deren Nullstelle gefunden werden soll, f' bezeichnet die Ableitung dieser Funktion. Der Startwert x_0 des Iterationsverfahrens kann unter der Einschränkung $f'(x_0) \neq 0$ (und weiteren, hier nicht genannten Einschränkungen) beliebig gewählt werden.

a) Schreiben Sie eine Funktion

```
newton.raphson <- function(f, x0, iter){
...
...
...
}
```

welche die Nullstelle der Funktion `f` nach einer vorgegebenen Anzahl an Iterationsschritten `iter`, beginnend mit dem Startwert `x0`, zurückgibt.
Machen Sie sich mit der R-Funktion `deriv` aus dem Paket `stats` vertraut und verwenden Sie sie an geeigneter Stelle.

b) Berechnen Sie die Nullstelle der Funktion $f(x) = 3 \cdot x^2 - 2 \cdot \log(25)$ mit 10 Iterationsschritten. Plotten Sie die Funktion und zeichnen Sie die berechnete Nullstelle in die Grafik ein.

c) Berechnen Sie die Nullstelle der Funktion $f(x) = (x-2)^2$ mit 5 Iterationsschritten. Plotten Sie die Funktion und zeichnen Sie die berechnete Nullstelle in die Grafik ein.

Aufgabe 5 (Simulation II) [30 Punkte]

Es existiert ein Verfahren, das eine auf dem *offenen* Intervall $(0, 1)$ gleichverteilte Zufallsvariable U verwendet um Zufallszahlen aus einer beliebigen Verteilungsfunktion F zu generieren. Dafür wird ausgenutzt, dass die Zufallsvariable $X := F^{-1}(U)$ der Verteilung F folgt.

Schreiben Sie eine Funktion

```
zufallszahl <- function(n, ...){
...
...
...
}
```

welche `n` Zufallszahlen mit dem oben beschriebenen Verfahren zur Verteilungsfunktion

$$F(x) = \begin{cases} 1 - \exp(-\vartheta x) & \text{, falls } x \geq 0 \\ 0 & \text{, sonst} \end{cases}$$

generiert und ausgibt. Erstellen Sie eine Grafik der geschätzten Dichte von $n = 10000$ erzeugten Zufallszahlen. Achten Sie darauf, dass Ihre Ergebnisse reproduzierbar sind.