Data Science mit R – Bike Sharing Demand
Projekt-Leitfaden

Lukas Jürgensmeier
Benjamin Lucht
Joel Tecle

Sommersemester 2019

Inhalt

Wofür ist dieses Dokument gedacht?

Herzlich willkommen in dem Projekt-Leitfaden für dein TechAcademy *Data Science mit R* Projekt!

Diese Kurzbeschreibung des Projektes soll dir erste Anhaltspunkte dafür geben, wie du zu einem Ergebnis kommst. Dieses Dokument ist jedoch bewusst keine Schritt-für-Schritt Anleitung, wie du das Projekt durchführen sollst. Uns ist es wichtig, dass du dich in deinem Team selbst mit der Aufgabenstellung beschäftigst und eigene Wege findest, wie du zu einem Ergebnis kommst.

Da es aber besonders am Anfang nicht ganz offensichtlich sein kann, welche Schritte du durchlaufen sollst, geben wir dir mit diesem Dokument eine kleine Hilfestellung. Es wird sehr oft vorkommen, dass du nicht weiter weißt. Das ist ganz normal und gehört zu dem Lernprozess dazu. Du findest in diesem Dokument Links zu sehr nützlichen Websites, wo Deine Frage vermutlich schon einmal beantwortet wurde. Falls auch googlen dich nicht weiter bringt, stehen dir natürlich die Mentoren per Slack und bei unseren Coding Meetups persönlich zur Verfügung.

Bevor es los geht

Um was geht das Projekt?

Ihr kennt es hier in Frankfurt: In der ganzen Stadt stehen Leihfarräder herum und warten darauf, ausgeliehen zu werden. Dieses Angebot ist nicht nur praktisch, sondern generiert auch große Mengen an Daten. Wir werden in diesem Semester einen detaillierten Datensatz der Firma Capital Bikeshare aus Washington, DC auswerten. Dabei ist die Analyse in zwei große Abschnitte aufgeteilt, die du nacheinander bearbeiten wirst.

Was ist das Ziel?

Explorative Datenanalyse – Lerne den Datensatz kennen

Als ersten Schritt werden wir den Datansatz *deskriptiv* kennen lernen. Das heißt, wir nähern uns dem Ziel, indem wir die Daten *beschreiben*. Bei Data Science Projekten ist es sehr wichtig, sich zu aller erst mit dem Datansatz vertraut zu machen. Welche Variablen sind in dem Datensatz enthalten und wie stehen sie im Verhältnis zueinander? Diese Fragen kann man sich sehr gut mit Grafiken beantworten.
Wir stellen dir dafür eine Reihe von strukturierten Aufgaben, die du nacheinander bearbeiten wirst. Anfänger, die bisher noch keine oder sehr wenige Statistik-Kenntnisse haben, können an diesem Punkt aufhören. Jedoch wird es gerade danach spannend. Versuche dich also auf jeden Fall trotzdem daran, wenn du noch etwas dazu lernen willst.

Nachfrage-Prognose – Wende statistische Methoden an

Dieser Part ist vornehmlich für etwas Fortgeschrittenere Teilnehmer vorgesehen. Wenn du

jedoch als Anfänger gut durch den ersten Abschnitt gekommen bist, empfehlen wir dir ausdrücklich, auch diesen Teil zu bearbeiten. Statistische Modelle sind ein enorm wichtiger Teil von Data Science.
Nachdem wir den Datensatz kennen gelernt haben, können wir in diesem Schritt ein Modell entwickeln, mit dem wir die Nachfrage nach Leihfahrrädern vorhersagen können. Ziel ist es, deine Vorhersage letztendlich bei Kaggle einzureichen. Kaggle ist eine Plattform von Google, die regelmäßig "Competitions" startet, bei denen es darum geht etwas besonders gut vorherzusagen. Du bekommst einen Trainings-Datensatz mit dem du deine Modelle trainierst. Danach wendest du dein Modell auf einen seperaten Test-Datensatz an. Kaggle überprüft dann die Genauigkeit deiner Vorhersagen und rankt deine Lösung danach.

Explorative Datenanalyse – Lerne den Datensatz kennen

1. Plot des Ausleihverhaltens über die Zeit (train data set)

Importiere den Datensatz in deinen Workspace. Die Daten sind bereits in deinem Projekt geladen. Da wir später zwischen Test- und Trainings-Datensätzen unterscheiden werden, nennen wir das Objekt, in dem wir den Datensatz speichern *train*.

```
train <- read.csv("BikeSharing_data.csv")
```

Verschaffe dir nun einen Überblick über den Datensatz. Dafür kannst du zum Beispiel folgende Funktionen nutzen:

```
head(train)
str(train)
summary(train)
```

Bevor du dir überlegen kannst, welche statistischen Methoden für die Vorhersage geeignet sein könnten, ist es immer nützlich sich die Zusammenhänge zwischen den Variablen visuell darzustellen. Diesen Prozess nenn man Exploratory Data Analysis (EDA).

Bevor wir damit starten können, müssen wir den Datensatz zuerst ein wenig bearbeiten, damit die Funktionen diesen verarbeiten können. Das klingt einfacher als gedacht – sehr oft sind die Daten in einem unbrauchbaren Format. In unserem Fall ist die Zeit-Variable *date.time* als string-Variable gespeichert. Diese müssen wir erst einmal in eine Datums-Variable umwandeln. Sehr praktisch dafür ist das package *lubridate*. Damit kannst du die Datums-Variablen einfach transformieren. Auch ist es sinnvoll, kategoriale Variablen, welche aktuell im *string*-Format sind, in einen *factor* zu transformieren. Behalte im Hinterkopf, welche Transformationen du durchgeführt hast, da wir diese später noch einmal für einen anderen Datensatz durchführen müssen.

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 3.5.1
```

```
train$datetime <- ymd_hms(train$datetime)
```

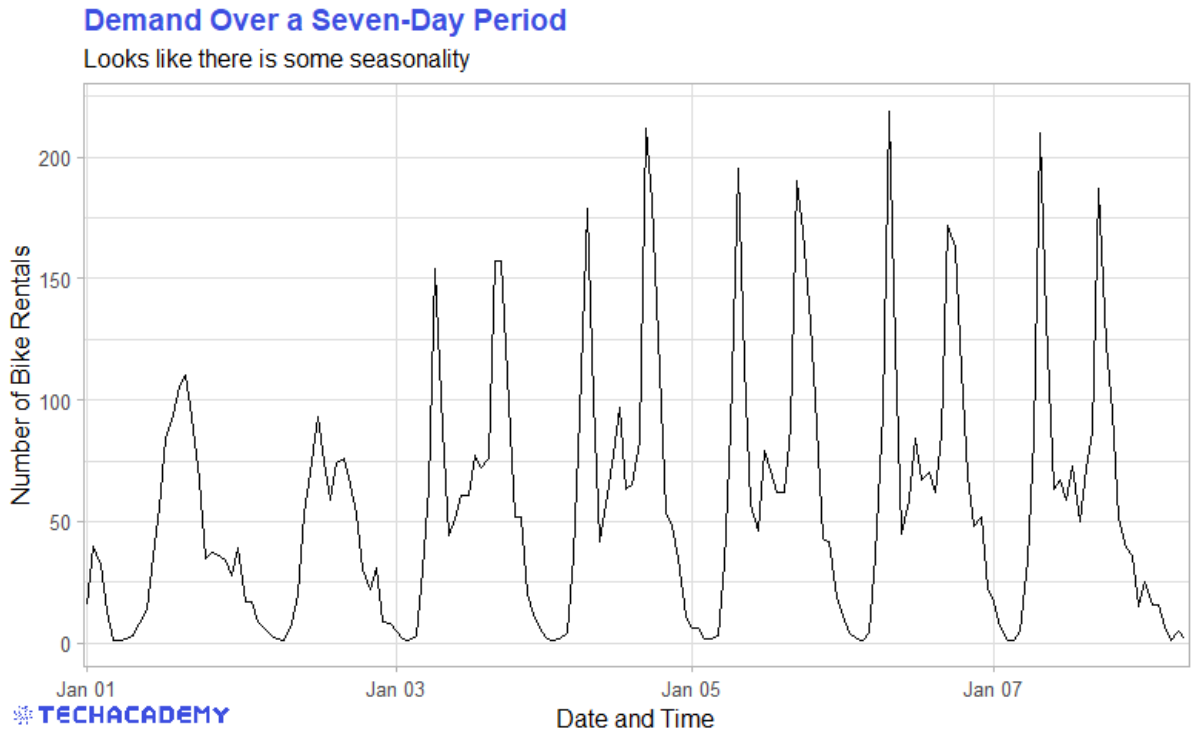
Wie man sehen kann gibt es eine Zeile pro Stunde und die Variable *count*, die Anzahl der in dieser Stunde ausgeliehenen Räder. Ziel ist es jetzt die Variable *count* über den Gesamtverlauf aller 10885 Beobachtungen zu plotten. Essentiell dafür ist das package *ggplot2*, von dem du auch schon in den DataCamp Kursen gehört hast. Damit kannst du alle möglichen Plots von einfachen Scatterplots über Boxplots und Histogramme bis hin zu fancy Grafiken wie Violin-Plots erstellen. Starte für's Erste mit einem Scatterplot.

```
library(ggplot2)
```

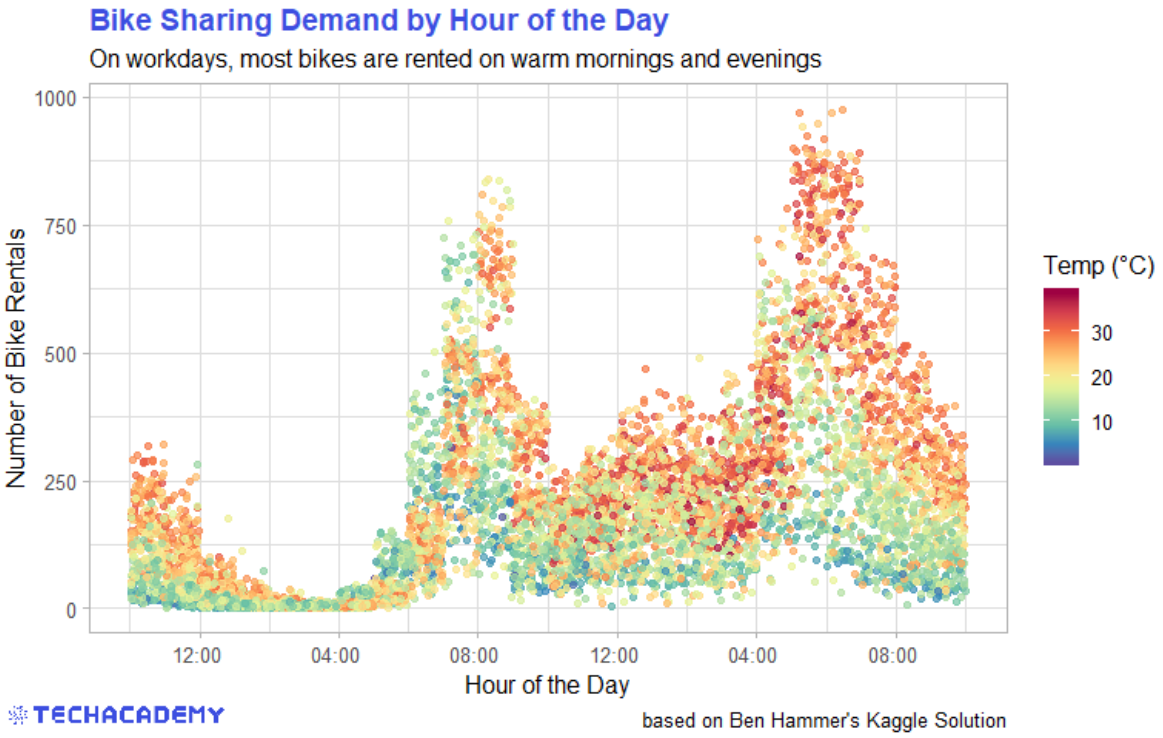
```
## Warning: package 'ggplot2' was built under R version 3.5.1
```

Plotte damit die Anzahl der ausgeliehenen Räder (*count*) gegen die Zeit. Kannst du daraus schon etwas ablesen?

Wähle im nächsten Schritt einen beliebigen 7-Tage Zeitraum und visualisiere diesen in einem Lineplot (Tipp: Starte z.B. bei bei der ersten Beobachtung). Was kann man beobachten? So in etwa könnte dein Plot aussehen:



Deiner Kreativität und deinen Design-Skills sind mit *ggplot2* keine Grenzen gesetzt. Ein Bespiel, wie man die einfachen Plots anpassen kann, ist diese Visualisierung des Zusammenhangs zwischen der Nachfrage nach Leihrädern nach Tageszeit sowie der Außentemperatur. Aber keine Angst, solch ein detaillierter Plot wird nicht von dir erwartet. Es soll nur zeigen, zu welchen Visualisierungen *ggplot2*() in der Lage ist.

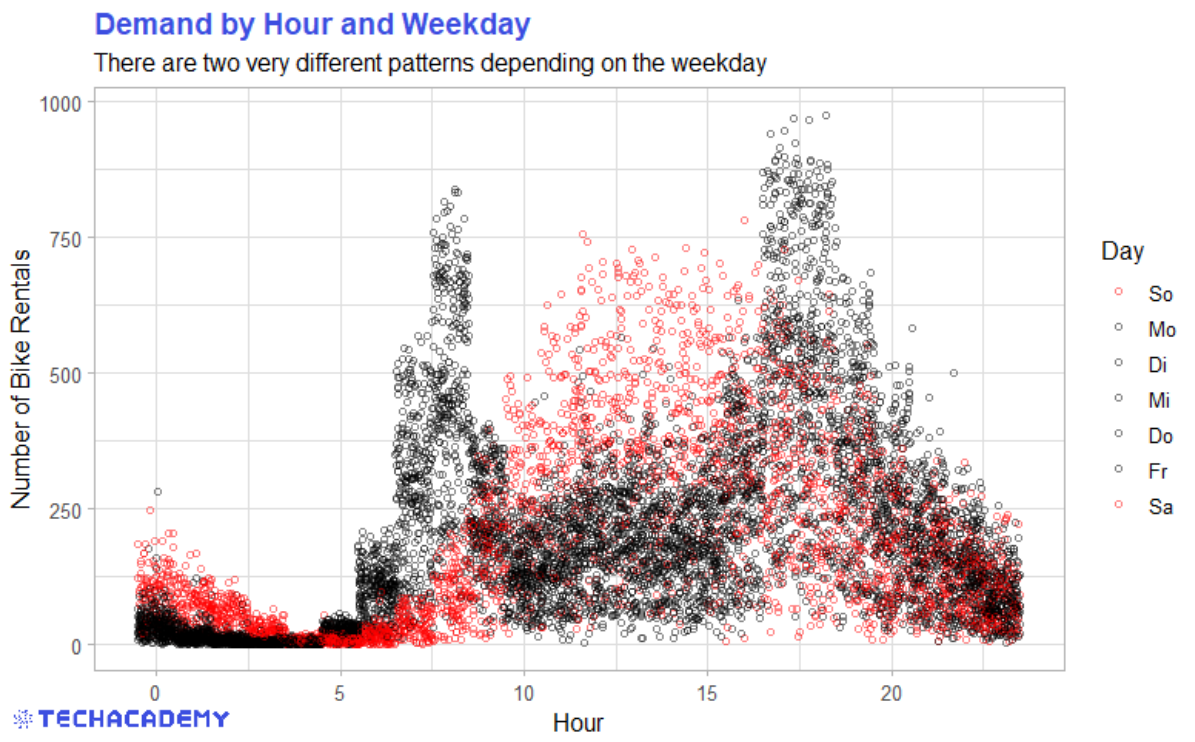


2. Ausleihverhalten nach Wochentag (train data set)

Wie in Aufgabe 1 festgestellt, scheint die Nachfrage nach Leihrädern nicht nur von der Uhrzeit, sondern auch vom Wochentag abzuhängen. Wie unterscheidet sich das Ausleihverhalten zwischen den Wochentagen, insbesondere im Hinblick auf die Uhrzeiten zu denen geliehen wird? Erstelle einen Line Plot mit der Summe der geliehenen Räder in Abhängigkeit der Uhrzeit. Erstelle danach den selben Plot (Summe der Räder gegen Uhrzeit), aber diesmal zusätzlich differenziert nach Wochentag. Was lässt sich hier feststellen?

Erster Schritt: In unserem Datensatz *train* befindet sich die Zeitangabe in der Variable *datetime*. Wenn in der ersten Aufgabe alles funktioniert hat, ist diese sogar im *Date*-Format. Finde aus dieser den Wochentag heraus. Tipp: Nutze das schon vorher verwendete Package. Darin findet sich auch eine Funktion, mit der man den Wochentag herausfinden kann.

Mit welcher Art von Plot du diese Aufgabe umsetzt, ist ganz dir überlassen. Probiere gerne mehrere verschiedene Arten (Line, Point, Boxplot) aus und entscheide welcher Plot den Zusammenhang am besten visualisiert. Hier ein Beispiel für einen Scatterplot, ähnlich zu dem vorherigen Plot:

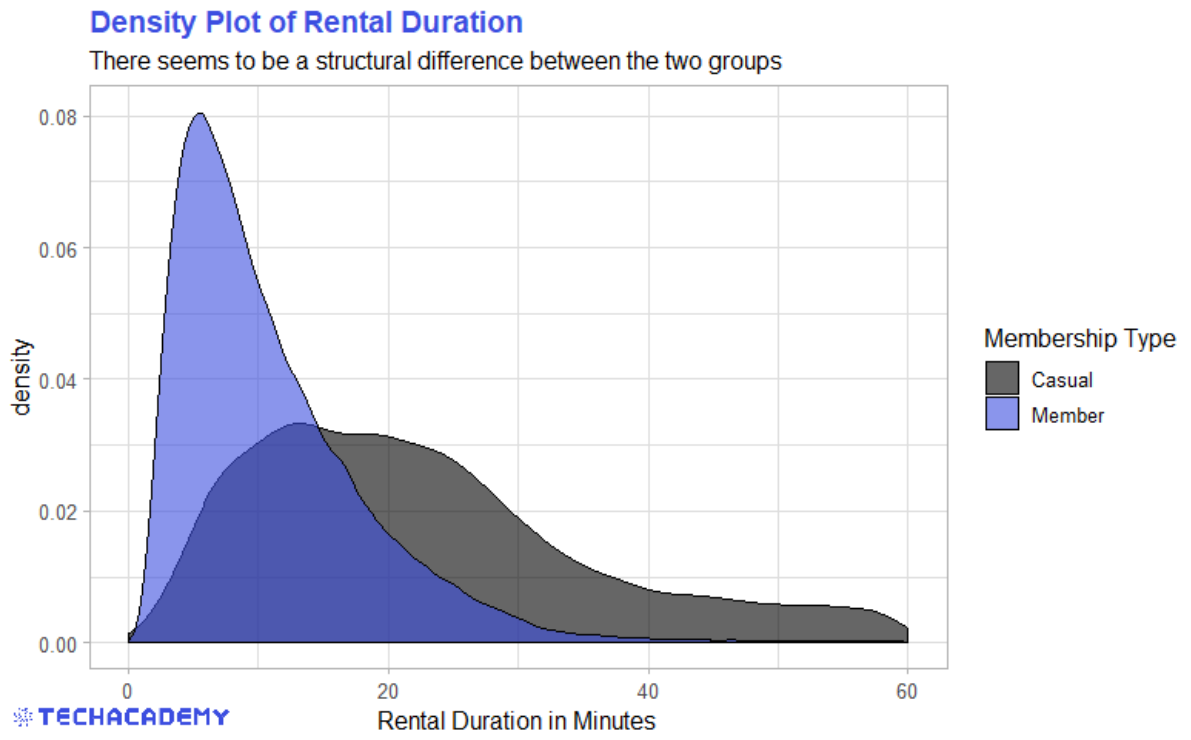


3. Wie lange werden die Räder ausgeliehen? (data_month)

Wir wechseln nun den Datensatz. Im vorherigen Kaggle-Datensatz waren die Anzahl der ausgeliehenen Fahrräder aggregiert nach Stunden. Auf der Website von Capital Bikeshare finden sich aber auch Datensätze auf einzelner Ausleih-Basis. Das heißt, jede einzelne Fahrt mit einem Fahrrad hat einen eigenen Eintrag in dem Datensatz. Wir haben beispielsweise die Daten von Februar 2019 schon in deinem Projekt geladen. Importiere diese Daten nun in deinen Workspace.

```
monthly <- read.csv("BikeSharing_data_201902.csv")
```

Schaue dir nun mit den schon bekannten Funktionen die Struktur des Datensatzes an. Generiere die typischen Lagemaße für die einzelnen Variablen (mean, median, standard deviation, etc.). Was fällt dir dabei auf? In welcher Einheit ist die Ausleihzeit angegeben? In diesem Datensatz finden sich auch Angaben über die Mitgliedschaft der Kunden bei dem Bike Sharing Anbieter. *Member.type* kann entweder *Member* oder *Casual* annehmen. Je nach Ausprägung dieser Variable ist der Kunde registriert oder nicht. Beschreibe die Ausleihzeiten je nach Art der Mitgliedschaft visuell.
Zum Beispiel kannst du dies in einem Density-Plot (siehe unten) darstellen. Probiere aber gerne wieder mehrere verschiedene Arten aus!
Tipp: in dem Datensatz befinden sich einige Ausreißer, die das Plotten schwierig gestalten. Überlege dir einen geeigneten Weg, damit umzugehen – sonst ist es schwer, einen lesbaren Graph zu erstellen.



4. Mergen & Karten zeichnen (data_month)

4.1 Datensätze mergen

Kommen wir zu einer etwas unangenehmeren Aufgabe, die aber zu fast jedem Data Science Projekt dazu gehört: Wir haben zwei Datensätze und müssen diese kombinieren. In einem weiteren Datensatz finden wir die Koordinaten der einzelnen Bike-Sharing-Stationen ("Bike-Sharing_stations.csv"). Lade diesen ebenfalls in deinen Workspace.

Schaue dir den neuen *stations*-Datensatz mit den bekannten Funktionen an. Im Datensatz *monthly* haben wir bereits für jede Fahrt eine Start- und End-Station angegeben. Für diese beiden Stationen wollen wir nun aus *Stations* die jeweiligen Start- und End-Koordinaten herausfinen und an *monthly* als neue Variablen anfügen.
Es gibt mehrere Lösungsmöglichkeiten für dieses Problem. Tipp: in beiden Datensätzen wird die Station einer eindeutigen Nummer zugewiesen. Über diesen kannst du die zwei Datensätze mergen. Dafür kommt zum Beispiel die Funktion *merge*() in Frage. Du wirst eventuell die Variablennamen umbenennen müssen. Dies funktioniert mit der *rename*()-Funktion aus dem *dplyr* package.

4.2 Koordinaten-Einträge mit Google Maps visualisieren (Bonus-Aufgabe)

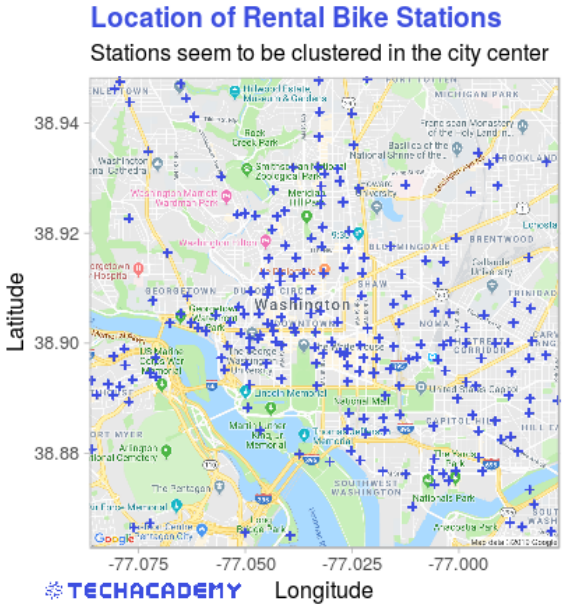
Dank deiner "merge-Skills" haben wir nun einen schönen Datensatz mit den Start- und End-Koordinaten zu jeder einzelnen Fahrt mit den Leihfahrrädern. Dieser Teil soll dir zeigen, was du nun mit Koordinaten-Daten anfangen kannst. Du kannst mit R über Googles API Karten von Google Maps in R importieren. Darauf kannst du dann Koordinaten einzeichnen und damit deinen Location-Datensatz visualisieren. Die Konfiguration der API ist etwas mühsam

Table 1: Mindestens diese Variablen sollte der gemergte Datensatz enthalten

Statistic	N	Mean	St. Dev.	Min	Pctl(25)	Pctl(75)	Max
End.station.number	100,558	31,346.920	226.996	31,000	31,209	31,604	32,407
Start.station.number	100,558	31,348.550	228.771	31,000	31,205	31,604	32,407
Duration	100,558	14.819	40.076	1.000	5.783	15.633	1,435.000
Start.lon	100,558	−77.032	0.030	−77.368	−77.044	−77.014	−76.826
Start.lat	100,558	38.906	0.023	38.783	38.895	38.915	39.124
End.lon	100,558	−77.031	0.030	−77.368	−77.044	−77.014	−76.826
End.lat	100,558	38.904	0.023	38.783	38.895	38.913	39.124

und man muss eine Kreditkarte bei Google hinterlegen (diese wird jedoch nicht belastet – der Service ist kostenlos). Deshalb musst du diese Aufgabe nicht bearbeiten – jedoch bekommst du damit sehr spannende Visualisierungen. Das ist aber ein tolles Beispiel, wie vielseitig und flexibel R einsetzbar ist.

Unten siehst du ein einfaches Beispiel, wie man die Ausleihstationen auf einer Karte der Stadt Washington, DC einzeichnen kann. Für diesen Plot wurden die packages *ggmap* sowie *mapsapi* mit der Funktion *register_google()* verwendet. Eine erste Karte in R zu laden ist nicht ganz intuitiv. Google dich selbständig durch die verschiedenen Foren und Blogs, bis du zu einem ersten Ergebnis kommst. Besonders musst du darauf achten, einen sogenannten API key für den Zugriff auf die Kartendaten zu generieren und diesen dann in *register_google()* zu definieren.



Du kannst auch weiter gehen und zum Beispiel die einzelnen Fahrten einzeichnen. Oder sogar die Dichte der einzelnen Fahrten auf einer Heat-Map visualisieren.

Herzlichen Glückwunsch – Du hast jetzt dank vieler Grafiken ein gutes Grundverständnis des Bike Sharing-Geschäfts. Damit hast du den ersten Teil des Projektes erfolgreich abgeschlossen! Wenn du in der Anfänger-Gruppe bist, sind deine Mindestvoraussetzungen hiermit erfüllt. Wir empfehlen dir aber trotzdem dringend, dich auch mit dem folgenden Teil auseinanderzusetzen.

Denn wir entwickeln jetzt Methoden, um die Zukunft vorauszusagen! Klingt spannend, oder?

Nachfrage-Prognose – Wende statistische Methoden an

Im vorherigen Teil hast du ein Gefühl für den Datensatz bekommen. Du weißt jetzt, welche Variablen enthalten sind und kennst ein paar charakteristische Eigenschaften des Datensatzes. Noch haben wir den Datensatz aber nur visualisiert. In diesem Abschnitt gehen wir einen Schritt weiter und wenden statistische Methoden an, um die Nachfrage nach Leihfahrrädern möglichst präzise vorherzusagen.

Um dein Modell am Schluss vergleichen zu können, nutzen wir die Online-Plattform Kaggle.

Erstelle dir einen kostenlosen Account bei Kaggle und gehe dann zu dem Bike Sharing Demand Projekt unter folgendem Link:

https://www.kaggle.com/c/bike-sharing-demand

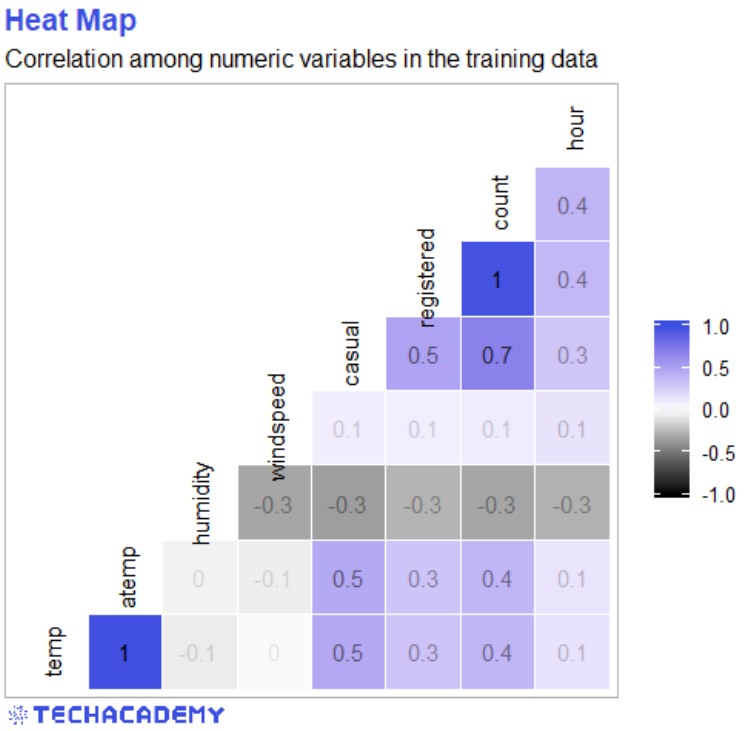
Du erhälst unter *Overview* eine detaillierte Einführung in das Projekt. Lese dir die Beschreibungen aufmerksam durch. Unter dem Reiter *Kernels* findest du einige (sehr fortgeschrittene) Lösungen zu dem Projekt. Wenn du einmal nicht weiter kommen solltest, kannst du dir Inspirationen davon holen. Aber bitte copy und paste nicht einfach eine Lösung davon. Das bringt dich persönlich nicht weiter.

1. Untersuche den Zusammenhang zwischen den Variablen näher

Wie stehen einzelne Variablen miteinander in Verbindung? Sprich inwiefern korrelieren die Variablen des Datensatzes miteinander? Das herauszufinden ist enorm wichtig für die Entscheidung, welches Modell du später anwenden kannst.
Ein guter Anfang ist es, eine Korrelationsmatrix zu erstellen. Du kannst das einfach mit dem *base* package und der Funktion *cor*() erstellen.

```
cor(train$temp, train$count)
```

Oder du visualisierst die Korrelationen in einer Heat-Map, wie nachfolgend dargestellt. Dafür ist zum Beispiel das package *ggcorr* sehr gut geeignet.



Aus diesem Graph kannst du schon sehr viel herauslesen. Scheinbar variiert die Nachfrage nach Fahrrädern sehr deutlich abhängig von der Tageszeit, sowie von der Temperatur.

2. Teste verschiedene Modelle und deren Qualität

Jetzt kannst du dich mit deinen Statistik-Kenntnissen austoben: Du brauchst jetzt ein Verfahren, wie du die Nachfrage nach Leihrädern zu einer bestimmten Zeit vorhersagen kannst.
Eine erste sehr einfache Herangehensweise wäre den Durchschnitt der Nachfrage als erste Vorhersage zu verwenden. Ziemlich sicher ist das jedoch nicht die beste Vorhersage.

Schon einmal was von einer linearen Regression gehört? Das wäre schon einmal ein deutlich besserer Ansatz. Jetzt kannst du deine Statistik-Skills ausspielen. Stelle doch einmal ein Modell mit der abhängigen Variable *count* auf. Welche unabhängigen Variablen könnten die Nachfrage nach Leihrädern erklären? Hat das Wetter etwas mit der Nachfrage zu tun? Leihen mehr Personen Fahrräder unter der Woche oder am Wochenende aus?

Starte mit einem ganz einfachen Modell mit nur einer erklärenden Variable. Zum Beispiel so:

$$count = \beta_0 + \beta_1 temp + \epsilon$$

In R kannst du eine einfache lineare Regression mit der Funktion $lm()$ implementieren. Die Ergebnisse davon gibst du dann mit der $summary()$ Funktion aus.

```
model <- lm(count ~ temp, data = train)
summary(model)
```

Hat die Temperatur einen statistisch signifikanten Einfluss auf die Nachfrage nach Leihfahrrädern? Vermutlich ja.

Wenn wir jedoch bei diesem sehr vereinfachten Modell bleiben, begehen wir einen typischen Fehler: Den sogenannten Omitted Variable Bias (OVB). Grob vereinfacht gesprochen vernachlässigen wir (im Statistik-Jargon: kontrollieren nicht für) Variablen, die einen signifikanten Einfluss auf die abhängige Variable haben. Man könnte vermuten, dass die Tageszeit auch eine große Rolle bei der Nachfrage nach Fahrrädern spielt. Wenn wir diese also nicht mit aufnehmen, ist die Schätzung des Effektes der Temperatur verzerrt und damit schlecht zu gebrauchen.

Eine Lösungsmöglichkeit ist, die vernachlässigten Variablen einfach mit in das Modell aufzunehmen – wie praktisch, dass diese auch schon in dem Datensatz enthalten sind. Stellen wir also ein umfangreicheres Modell auf, das die Tageszeit mit aufnimmt:

$$count = \beta_0 + \beta_1 temp + \beta_2 hour + \epsilon$$

Hier siehst du das Ergebnis der beiden Modelle. Welches ist besser geeignet? Macht es Sinn, die Variable *hour* in dieser Form mit in das lineare Regressionsmodell aufzuführen?

Table 2: Lineare Regression – Modell-Vergleich

	Dependent variable:	
	count	
	(1)	(2)
temp	9.171***	7.985***
	(0.205)	(0.192)
hour		9.185***
		(0.216)
Constant	6.046	−75.973***
	(4.439)	(4.541)
Observations	10,886	10,886
R^2	0.156	0.276
Adjusted R^2	0.156	0.276
Residual Std. Error	166.464 (df = 10884)	154.152 (df = 10883)
F Statistic	2,005.529*** (df = 1; 10884)	2,073.866*** (df = 2; 10883)
Note:		*p<0.1; **p<0.05; ***p<0.01

Tipp: Solch eine Tabelle kannst du relativ einfach mit dem package *stargazer* erstellen.

Jetzt bist du dran: Probiere mehrere Kombinationen aus und vergleiche die Ergebnisse deiner Modelle. Hinterfrage dabei immer kritisch, ob es Sinn macht, eine jeweilige Variable in das Modell mit aufzunehmen und was das für die Qualität deines Modells bedeutet.

Hier sind deiner Kreativität keine Grenze gesetzt. Du musst nicht bei einer einfachen linearen Regression bleiben. Gibt es nicht-lineare Zusammenhänge? Wie geht man mit unrealistischen Vorhersagen um? Warum nicht einmal ein künstliches Neuronales Netz mit dem Regressionsmodell vergleichen?

3. Von Training zu Testen – Treffe Vorhersagen

Jetzt hast du dein Modell ausgewählt und mit dem Trainings-Datensatz *trainiert*. Doch wie gut geht das Modell mit Daten um, die es noch nicht gesehen hat? Das ist ein sehr wichtiger Test für die Qualität deines Modells.

Hat dein Modell nur die vorhandenen Muster im Trainings-Datensatz "auswendig" gelernt? Dann wären die Zusammenhänge aus dem Trainings-Datensatz nicht übertragbar auf den Test-Datensatz. Beim sogenannten overfitting hat das Modell zu nah am Trainings-Datensatz gelernt und liefert deshalb schlechte Vorhersagen bei unbekannten Daten – in deinem Test-Datensatz. Auf der andern Seite gibt es auch das Problem underfitting: Dein Modell hat die tatsächlichen Zusammenhänge der Daten nicht gelernt und sagt deshalb in dem Test-Datensatz schlecht voraus. Es gilt also die goldene Mitte zwischen den beiden Problemen zu finden.

Jetzt wird die Unterscheidung zwischen Trainings- und Testdatensatz wichtig. Wir nutzen *train*, um ein Modell zu *trainieren* und *test*, um die Qualität unseres Modells letztendlich zu *testen*. Wir haben bereits die benötigten Daten in deinem Workspace geladen.

Lade nun zusätzlich zu dem Datensatz *train*, den du bereits vorher verwendet hast, den Datensatz *test*:

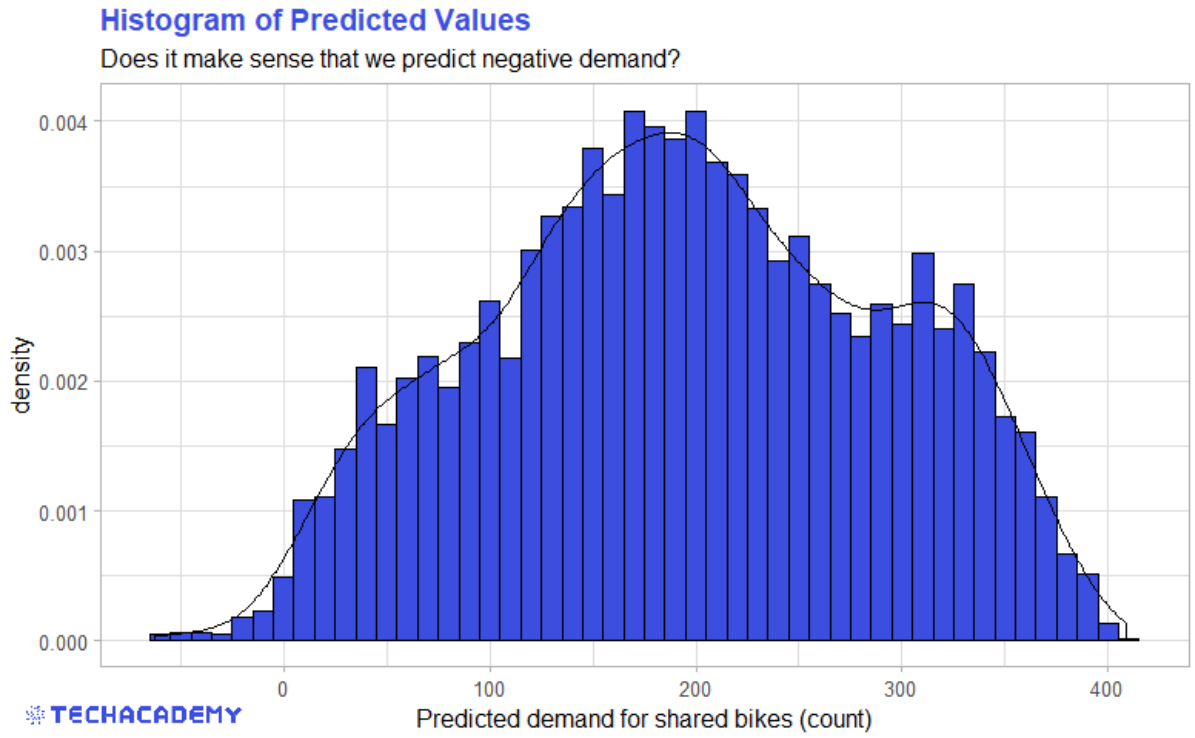
```
test <- read.csv("BikeSharing_data_test.csv")
```

Erinnerst du dich an die Transformationen, welche du auf den *test* Datensatz angewandt hast? Denke daran, diese Transformationen noch einmal für *test* durchzuführen, da du später deine Methoden auch darauf anwenden wirst und dies nur bei identisch transformierten Variablen in beiden Datensätzen funktioniert.

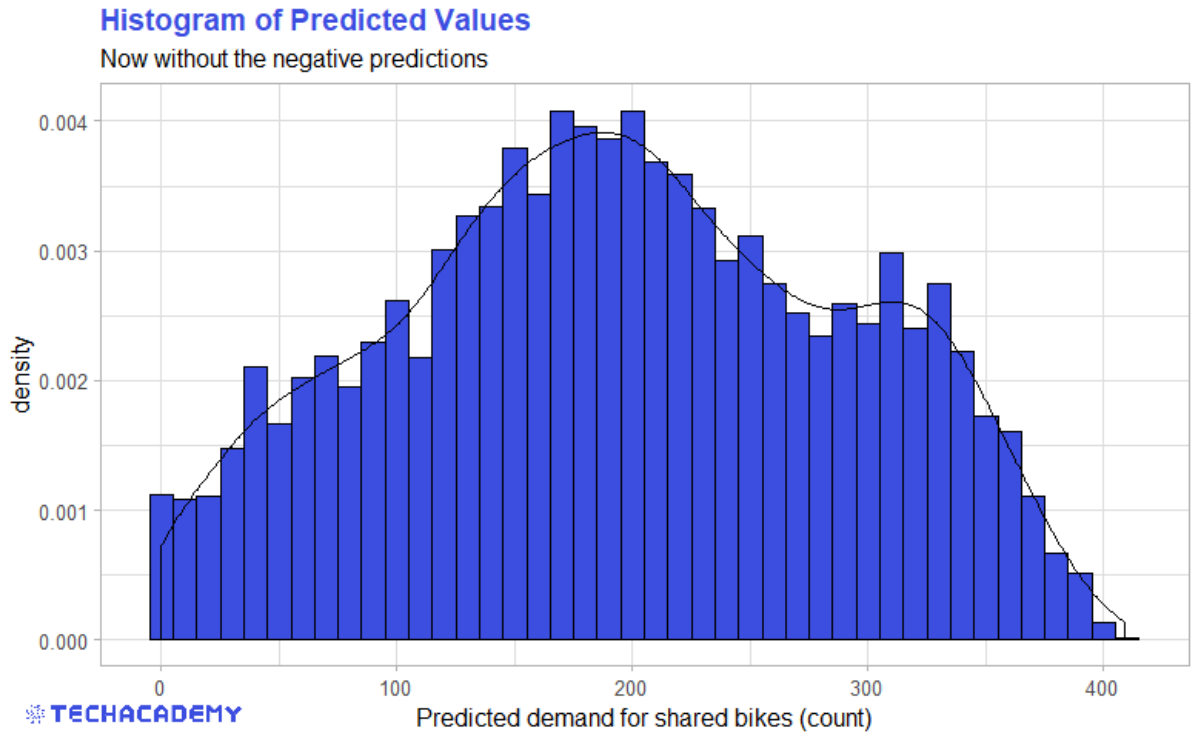
Jetzt heißt es, dein favorisiertes Modell auf den Test-Datensatz anzuwenden. Dabei ist die Funktion $predict()$ sehr hilfreich. Diese gibt einen Vektor von allen Vorhersagen eines bestimmten Modells aus. Hier kannst du jetzt spezifizieren, mit welchem Datensatz das ganze geschehen soll. In diesem Fall ist das der Test-Datensatz.

Im zweiten Schritt erstellst du dann einen neuen Datensatz, der sowohl die Datums-Variable als auch unsere Vorhersage enthält. Das klappt zum Beispiel mit der Funktion $data.frame()$. Ziel ist es, die Variable *count* für den *test* Datensatz mit dem Modell aus dem *train* Datensatz vorherzusagen.

Danach kannst du wieder einen Schritt zurück zum Visualisieren gehen. Überprüfe, ob die Vorhersagen valide sind. Mit einem Histogramm kannst du beispielsweise herausfinden, wie die Verteilung deiner Vorhersagen ist und ob unrealistische Werte vorhergesagt werden.



In diesem Histogramm sehen wir die Verteilung der Vorhersagen aus einem sehr simplen Regressionsmodell: Auffällig ist, dass wir negative Nachfrage nach Leihfarrädern vorhergesagt haben. Macht das Sinn? Mit Grafiken kannst du relativ einfach überprüfen, ob deine Ergebnisse Sinn machen. Wir können nun unser Modell weiter anpassen, um diesen "Fehler" zu vermeiden. Die einfachste Möglichkeit ist, alle negativen Werte durch 0 zu ersetzen. Das ist aber ziemlich sicher nicht die beste Methode! Findest du einen sinnvolleren Weg, keine negativen Werte vorherzusagen?



4. Lade den Datensatz auf Kaggle hoch

Wenn du mit deinem Modell und den Vorhersagen zufrieden bist, lade den Datensatz auf Kaggle hoch. Du siehst dann gleich, wie deine Lösung im Vergleich zu anderen Teilnehmern abschließt. Aber keine Angst vor einer schlechten Platzierung, auf Kaggle misst du dich mit extrem guten Data Scientists.

Zufrieden mit dem Ergebnis? Nein? Dann gehe noch einmal ein paar Schritte zurück und verfeinere dein Modell. Du kannst beliebig oft eine Lösung auf Kaggle hochladen.

Noch Fragen?

Du kommst nicht weiter? Oder willst dein Modell noch weiter verbessern, weißt aber nicht genau wie? Oder dir fällt gerade nicht ein, wie man etwas bestimmtes im Code umsetzt? Schaue dir noch einmal unser Handout an. Dort findest du hilfreiche Hinweise, wie du in diesem Fall weiter verfahren und wo du nach einer Lösung für deine Fragestellung suchen kannst.

Beschreibung der Variablen in den Datensätzen

Wenn du dir einmal nicht sicher bist, welche Variable was aussagt, kannst du hier die Beschreibung nachschauen.

Train und Test

datetime - hourly date + timestamp

season - 1 = spring, 2 = summer, 3 = fall, 4 = winter

holiday - whether the day is considered a holiday

workingday - whether the day is neither a weekend nor holiday

weather

1: Clear, Few clouds, Partly cloudy, Partly cloudy

2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp - temperature in Celsius

atemp - "feels like" temperature in Celsius

humidity - relative humidity

windspeed - wind speed

casual - number of non-registered user rentals initiated

registered - number of registered user rentals initiated

count - number of total rentals

data_month

Duration - amount of time a bike was rented in seconds

Startdate - start of the rental

Enddate - end of the rental

Startstationnumber - id of the station where bike rental started

Startstation - name of the location of this station

Endstationnumber - id of the station where bike rental ended

Endstation - name of the location of this station

Bikenumber - id of rented bike

Membertype - type of customer who rented the bike (‘Member’ or ‘Casual’)

Stations

Stationnumber - id of the station

Stationname - name of the location of this station

lon - longitude of the station

lat - latitude of the station