Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann

# Bericht zur Projektarbeit „Titanic“

## Klärung der Aufgabenstellung

**Ziel:** Aus gegebenen Informationen zu Passagieren der Titanic Vorhersagen zu deren Überleben treffen.

**Qualitätskriterien:** Genauigkeit der Vorhersage, „Score“ auf Testdaten. Hinweis: Externer Score bei Einreichung der Ergebnisse bei kaggle möglich.

**Überwachtes Lernen:** Label (überlebt ja / nein) sind vorhanden.

## Beschaffung der Daten

Die verwendeten Daten stammen von kaggle.com aus der Titanic-Competition. (Quelle: https://www.kaggle.com/c/titanic ). Sie scheinen so bzw. in ähnlicher Form auch auf anderen Internet-Seiten verfügbar zu sein. Der Download ist einfach und schnell. Es handelt sich um zwei gut verwendbare CSV-Dateien. Die Größe in Kilo-Byte sind 28 KB (die Testdaten ohne Label für die Competition) und ca. 60 KB (die Daten mit Label).

„train.csv“ hat 891 Zeilen und 12 Spalten, „test.csv“ hat 418 Zeilen und 11 Spalten.

**Hinweis:** Es sind aufgrund des Formats „Competition“ zwei Datensätze vorhanden. Wir stellen den Datensatz „test“ zurück und arbeiten von jetzt an mit „train“ und bezeichnen diesen Datensatz als „die (gegebenen) Daten“.

## Beschreibung der Daten

Die Daten sind als CSV Dateien mit 891 Zeilen und 12 Spalten gegeben.

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 | NaN | S |

Es gibt folgenden Spalten:

- **PassengerID**: Eine Nummer, die einem Passagier zugeordnet ist. Wie eindeutig diese Zuordnung ist, muss noch hinterfragt werden.
- **Survived**: Eine 0 oder eine 1 entsprechend der Information „überlebt 1“ und „nicht überlebt 0“.
- **Pclass:** Klasse - Von 1 zu 3 mit absteigender Qualität/Luxus/Preis
- **Name**: Name des Passagiers.
- **Sex**: Geschlecht des Passagiers.
- **Age**: Alter, als Bruch wenn jünger als ein Jahr; sonst ganze Jahre bzw. „XX.5“ deutet auf Schätzung.
- **SibSp**: Anzahl Geschwister und/oder Ehepartner; vermutlich ist die Anzahl der mitreisenden Geschwister und/oder Ehepartner gemeint. *Hinweis:* Evtl. später darauf noch genauer eingehen.
- **Parch**: Anzahl Elternteile und/oder Kinder; vermutlich ist die Anzahl der mitreisenden Elternteile und/oder Kinder gemeint. Mitreisende Kinderbetreuungspersonen (damals durfte man bestimmt noch Kinderfrau sagen) wurden hier nicht erwähnt. *Hinweis:* Evtl. später darauf noch genauer eingehen.
- **Ticket**: Bezeichner für das Ticket. Hier ist es noch ungenau, wie man den Ticket „lesen“ kann.
- **Fare**: Fahrpreis (insgesamt) … Hinweis: Wofür genau wird noch diskutiert werden!
- **Cabin**: Kabinenbezeichnung bestehen aus Deck und Cabinenummer.
- **Embarked**: in welchem Hafen eingestiegen (drei Häfen: S, C, Q)

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann

## Zielgröße

**Die Zielgröße ist die Spalte „Survived“.** Man kann später mit dieser Spalte überprüfen, ob jemanden überlebt hat.

## Vollständigkeit und Typ der Daten

Die Daten sind verschiedenen Typs und unterschiedlich vollständig. Mit Hilfe von Pandas.Dataframe.info() werden folgende Informationen geliefert:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

**Unvollständig** sind „Age“ mit 714 statt 891 Einträgen, „Cabin“ mit 204 statt 891 Einträgen und „Embarked“ mit 889 statt 891 Einträgen.

**Typ der Daten:** Es gibt numerische (Integer und Float) sowie nicht-numerische (object). Genauere Informationen über Pandas.Dataframe.describe():

| | PassengerId | Survived | Pclass | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|---|
| count | 891.000000 | 891.000000 | 891.000000 | 714.000000 | 891.000000 | 891.000000 | 891.000000 |
| mean | 446.000000 | 0.383838 | 2.308642 | 29.699118 | 0.523008 | 0.381594 | 32.204208 |
| std | 257.353842 | 0.486592 | 0.836071 | 14.526497 | 1.102743 | 0.806057 | 49.693429 |
| min | 1.000000 | 0.000000 | 1.000000 | 0.420000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 223.500000 | 0.000000 | 2.000000 | 20.125000 | 0.000000 | 0.000000 | 7.910400 |
| 50% | 446.000000 | 0.000000 | 3.000000 | 28.000000 | 0.000000 | 0.000000 | 14.454200 |
| 75% | 668.500000 | 1.000000 | 3.000000 | 38.000000 | 1.000000 | 0.000000 | 31.000000 |
| max | 891.000000 | 1.000000 | 3.000000 | 80.000000 | 8.000000 | 6.000000 | 512.329200 |

- **PassengerID** – fortlaufende Nummer von 1 bis 891
- **Survived** - 0 oder 1, median sagt: Mehr Menschen sind gestorben als überlebt haben.
- **Pclass –** 1, 2, 3
- **Age** – von 0.42 bis 80.0
- **SibSp** – von 0 bis 8
- **Parch** – von 0 bis 6
- **Ticket** – Kombination Zahlen und Buchstaben
- **Fare** – von 0 bis 512.3292 (Einheit, Nachkomma?)
- **Cabin** – Kombination von Buchstaben und Zahlen
- **Embarked** – C = Cherbourg, Q = Queenstown, S = Southampton

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann

## Wichtigkeit der Features – Vorabdiskussion

Wir haben zu Beginn diskutiert welche **Features wichtig** sein dürften. Recherche: Die Kollision war um 23:40, also waren die Passagiere bei der Abendunterhaltung bzw. bereits in den Kabinen.

- Pclass: Niedrigere Klasse (erste Klasse „1“) haben eventuell bessere Lage zu den Rettungsbooten
- Fare: Zusammenhang Pclass – je höher die Klasse, desto mehr ist die Fare.
- Cabin: ebenfalls Zshg Pclass, Fare, wegen großer Unvollständigkeit wurde dieses Feature aber zurückgestellt
- Age und/oder Sex: „Frauen und Kinder zuerst“, wobei hier diskutiert wurde, ob dies wirklich stattgefunden ist.

Als eher unwichtig haben wir eingestuft

- Embarked
- Name – Es soll erst einmal egal sein, wie die Person heißt
- PassengerID, den dies liefert keine relevante Information über den Passagier

Die Features „SibSp“ und „Parch“ haben wir genauer diskutiert: Welche Informationen enthalten sie und wie kann man diese sinnvoll ausnutzen?

## Diskussion der Features „SibSp“, „Parch“ und Fare

Um über die genau Aussage dieser Spalten zu bestimmen, wurde besondere Datensätze angeschaut:

Aus der vorherigen Analyse der Daten wissen wir, dass die jüngste Person in der Liste 0.42 Jahre alt ist:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **803** | 804 | 1 | 3 | Thomas, Master. Assad Alexander | male | 0.42 | 0 | 1 | 2625 | 8.5167 | NaN | C |

Passagier Master Assad Alexander Thomas hat SibSp 0, hat also weder Geschwister noch ist er verheiratet. Er hat 1 Parch, reist also mit einem Elternteil oder Kind. Als Nebenbemerkung: Das Baby ist der Ticketinhaber.

Interpretation A des Ticketpreises: „Fare“: dieser Ticketpreis bezieht sich auf 2 Personen; den Passagier selber + 1 Parch.

Die Suche nach dem Nachnamen „Thomas“ führt im Kaggle-Datensatz „train.csv“ zu keinem Ergebnis. Im Kaggle-Datensatz „test.csv“ findet man:

| | PassengerId | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **104** | 996 | 3 | Thomas, Mrs. Alexander (Thamine Thelma")" | female | 16.0 | 1 | 1 | 2625 | 8.5167 | NaN | C |
| **116** | 1008 | 3 | Thomas, Mr. John | male | NaN | 0 | 0 | 2681 | 6.4375 | NaN | C |
| **133** | 1025 | 3 | Thomas, Mr. Charles P | male | NaN | 1 | 0 | 2621 | 6.4375 | NaN | C |
| **332** | 1224 | 3 | Thomas, Mr. Tannous | male | NaN | 0 | 0 | 2684 | 7.2250 | NaN | C |

Also vier Personen, von denen nur eine Parch ungleich 0 hat, also mit einem Kind oder einem Elternteil reist. Mrs. Alexander Thamine Thelma Thomas ist 16, könnte also ein Baby haben. Auf jeden Fall hat sie die gleiche Ticketnummer wie Master Assad Alexander Thomas (ein Indiz). Darüber hinaus reist sie aber mit SibSp 1, also einem Ehepartner oder einem Geschwister. Wäre es ein Ehepartner, so wäre dieser vermutlich auch dem Baby zugeordnet (sonst wird so langsam ein echtes Familiendrama daraus). Also ist hier vermutlich ein Geschwister gemeint. Jedoch ist es unklar, ob mehrere Geschwister beinhaltet sind, die jedoch einen anderen Nachname haben.

Interpretation B des Ticketpreises: „Fare“: Dieser Ticketpreis bezieht sich auf 3 Personen; den Passagier selber + 1 Parch + 1 SibSp.

Es besteht eine Diskreptanz zwischen beide Interpretationen, deswegen wissen wir nicht genau, wie man mit dem Fare umgehen soll. Wir haben uns für folgendes Entschieden:

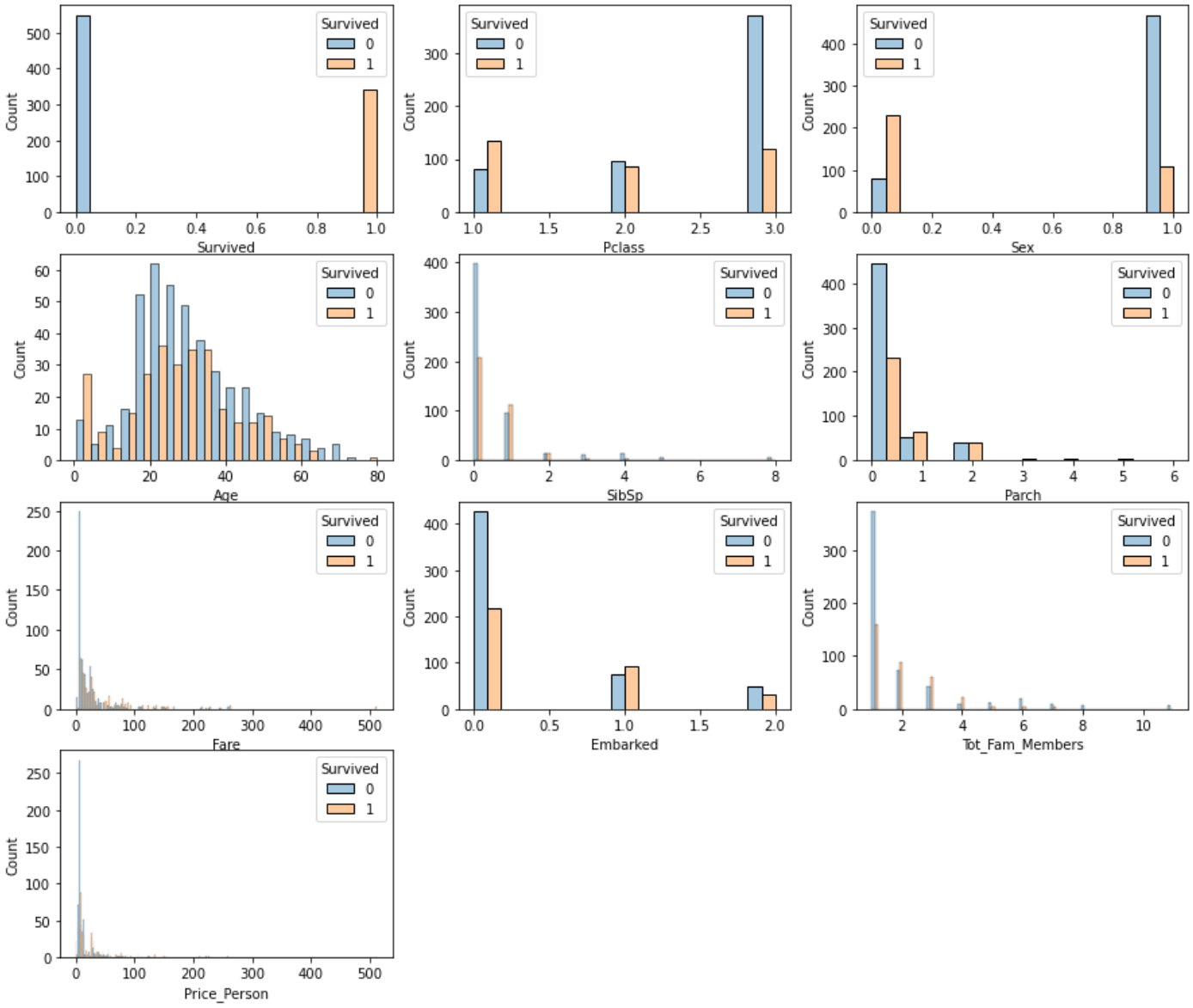
Um die Gesamtanzahl der fahrenden Personen pro „Fare“ zu bestimmen, haben wir die Anzahl der Personen in SibSp und Parch addieren und anschließend zusätzlich noch die Person dazu addiert, die das Ticket gekauft hat (neue Spalte: „Tot_Fam_Members“).

Weiterhin soll hier dann anschließend der Fare-Preis zwischen der Anzahl der Familienmitglieder geteilt werden, sodass man den „Preis_Person“ erhalten kann, also den Preis pro Person. Hier wird vernachlässigt, dass eventuell Kinder weniger zahlen müssen als Erwachsene.

## Visualisierung (Jupyter Notebook)

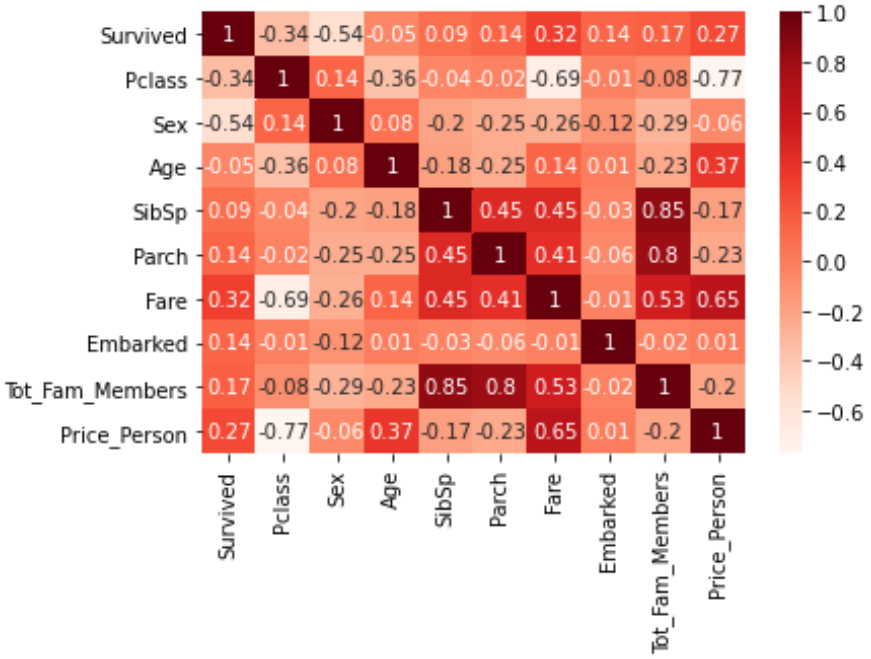
Histogramme der Überlebenden bzgl. aller numerischen Variablen wurden erstellt. Hier wurden die Hafen S, C, Q zu 1, 2, 3 geändert und das Geschlecht zu 0 und 1 (0: female, 1: male).

Weiterhin wurde nach der obigen Diskussion noch zwei weitere Spalten erstellt: Totale Anzahl von Familienmitgliedern und Preis pro Person. Hier wollten wird noch einige Zusammenhänge darstellen, um zu sehen, ob die Anzahl der Überlebten mit bestimmten Eigenschaften zusammenhängen.

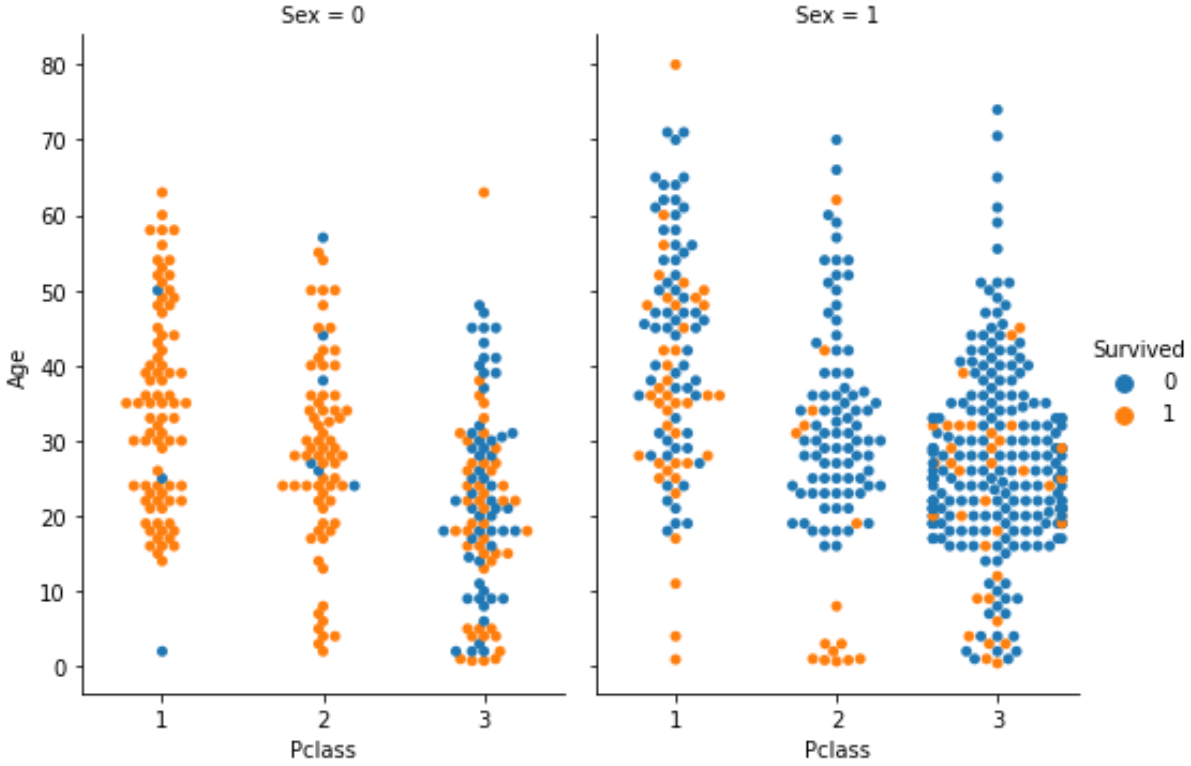



Mit Hilfe eines Heatmaps wurde ein Korrelationsschaubild hinzugefügt. Am relevantesten für das gesetzte Ziel ist die Korrelation der Spalte „Survived“ mit den anderen Spalten:

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann




Die Spalte „Survived“ hat die höchsten Korrelationen mit Pclass und Sex. Entsprechend wurde dies genauer betrachtet. Weiterhin haben wir entschieden, die Kinder in dieser Betrachtung miteinzubeziehen, um die Behauptung „Kinder und Frauen zuerst“ zu bestätigen:




Man kann aus dieser Grafik folgendes ablesen (Sex = 0 entspricht „Frau“, Sex = 1 entspricht „Mann“):

- Geschlecht: Frauen wurden häufiger gerettet als Männer (visuell mehr „orangene“ Punkte im linken Bild als im rechten). Das Alter scheint hier keine Rolle zu spielen.
- Klasse: In beiden Diagrammen sieht man, dass mit absteigender Klasse die Anzahl der nicht-gerettete Personen steigt.
- Alter: Insgesamt sieht man, dass in der ersten und zweiten Klasse fast alle Kinder (bis ca. 15 Jahre) gerettet wurden. In der dritten Klasse kann man dies nicht verallgemeinern.

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann
Daten umformen

# Daten Umformen

Nicht jede Spalte ist vollständig und es fehlen dementsprechend Daten. Bei folgenden Spalten wurden Berenigungen durchgeführt mit der Funktion „drop_dataset“:

- „Embarked“: Es wurden nur zwei Zeilen mit fehlenden Werten entfernt.
- „Cabin“: Die ganze Spalte wurde gelöscht, weil insgesamt 687 Einträge fehlen (ca. 75 % der Gesamtmenge).
- „Ticket“: Es fehlen zwar keine Daten, die Spalte wurde aufgrund fehlender Struktur der Einträge gelöscht (Einträge nur mit Nummern, oder Nummer und Buchstaben).
- „Name“: Spalte gelöscht wegen redundanten Informationen.
- „PassengerID“: Spalte gelöscht wegen redundanten Informationen.

Der Datensatz wurde stratifiziert in Trainingsdaten und Testdaten mit einem Verhältnis von 80:20. Wichtig ist, dass mit Hilfe des StratifiedShuffleSplit() die gleichmäßig mit dem Survived-Anteil getrennt wurden.

## Trennung in Train- und Test-Daten (Dateimengen):

Um mit den uns gegebenen Train-Daten zu arbeiten und da bei den von Kaggle gegebenen Test-Daten das Target („Survived“) fehlt, wurde der **Train-Datensatz, welcher als Gesamtdatensatz** behandelt und entsprechend „df“ benannt wurde, **in einen TRAIN_SET und TEST_SET gesplittet**.

```
strat = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in strat.split(df, df["Survived"]):
    train_set = df.iloc[train_index]
    test_set = df.iloc[test_index]
```

Hierbei wurde eine **stratifizierte Splittung** nach „**Survived“** vorgenommen. Eine Stratifikation ist hierfür zulässig, da es sich bei dieser Variable um eine kategorische Variable handelt und diese ebenfalls am „meisten“ (also 100%) mit der Traget-Variable korrelieren, da **sie selbst die Target-Variable** darstellt.

Eine Stratifizierung nach einer anderen Variable erschien uns nicht zielführend, denn wir brauchen einen prozentualen Anteil, bei dem gleichmäßig zwischen survived und nicht survived getrennt wurde.

Das test_set wurde zu **20% abgespalten**, da unser Datensatz „df“ groß genug (891 Samples) ist.

Im Anschluss wurden die Datensätze wie gewohnt in eine **X-Test und X-Train** (jeweils die Feature-Spalten) und **Y-Test und Y-Train** (jeweils die Targetspalte) aufgesplittet.

```
# xTrain, xTest, yTrain, yTest:
xTrain_unprepared = train_set.drop("Survived", axis=1)
yTrain = train_set["Survived"]
xTest_unprepared = test_set.drop("Survived", axis=1)
yTest = test_set["Survived"]
```

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann

### Umformung der Daten mit Pipelines und ColumnTransorfmer:

Im nächsten Schritt wurde mithilfe von zwei Pipeline folgende Spalten bearbeitet. Hier fand eine Trennung zwischen numerischen und kategorischen Daten statt.

- Die numerische Daten gingen durch die numerical_pipeline durch. Hier wurden mit Hilfe des KNNImputer() fehlende Datensätze hinzugefühlt – die meistbetroffene (und einzige Spalte) war die „Age"-Spalte. Wir wählten den KNNImputer(), denn bei der Mean- und Median-Methode (mit SimpleImputer()) wurden die Daten stärker verfälscht durch die Verschiebung des Gewichts der Daten. Durch den KNNImputer() eine größte Variabilität gewährleistet, denn hier richtet man sich nach dem Nachbarwerte der fehlenden Werte.
- Anschließend wurden in der numerical pipeline mit der Klasse CombinedAttributesAdder die Spalten Tot_Fam_Members und Price_PP erstellt. Anschließend wurden die numerischen Daten mit dem MinMaxScaler() durchgeführt.
- Man wählte den MinMaxScaler(), da wir sehr viele Daten mit Wert 1 haben. Diese würden alle zu 0 werden, wenn wir den StandardScaler nutzen würden.
- Die Spalte „Sex" wurde mit dem OrdinalEncoder() in Zahlen kategorisiert / kodiert.
- Die Spalte„Embarked" wurde mit OneHotEncoder() in einen Vektor mit drei Einträge kategorisiert / kodiert.

Die Zusammenführung fand mit dem ColumnTransformer() statt. Somit bestehen die Trainingsdaten aus 711 Zeilen und 11 Spalten und die Testdaten aus 178 Zeilen und 11 Spalten.

## Daten analysieren und Modelle optimieren

### Erste-Maschine Learning Modelle:

Zur ersten und schnellen Analyse der Daten wurden **sechs Algorithmen** genutzt. Wichtig war hierbei nochmals die Zielstellung, eine s**upervised Classification** durchzuführen. Entsprechend kamen nur **Classifier** zum Einsatz. Folgenden Algorithmen wurden ausgewählt mit den entsprechenden Begründungen:

| Algorithmus | Begründung für Nutzung |
|---|---|
| Gaussian-Naive Bayes | PRO:<br>- Schnelle Laufzeit<br>- Wenige Hyperparameter<br>- Uns bekannt und entsprechend wissen wir die Theorie dahinter<br>- Sehr leicht und intuitiv zu verwenden<br>KONTRA:<br>- Nicht gut einstellbar für spezifische Fälle |
| Logistische Regression | PRO:<br>- Schnelle Laufzeit<br>- Eine neue Methode ausprobieren<br>- Wenige Hyperparametern<br>- Intuitiv und leicht zu verwenden<br>KONTRA:<br>- Unbekannt<br>- Für uns nicht einschätzbar, wann sinnvoll einsetzbar |
| Kneighbors Classifier | PRO:<br>- Intuitiv und leicht zu verwenden<br>- Wenig Hyperparameter<br>- Schnelle Laufzeit<br>- Gut für kleine Datensätze (wie in diesem Fall)<br>KONTRA:<br>- Gleiche Wichtigkeit aller Feature wird angenommen: Ist einstellbar, uns aber nicht bekannt wie genau (zu wenig Erfahrungswerte) |

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann

| | |
|---|---|
| Support Vector Classification | PRO:<br>- Uns bekannt und entsprechend wissen wir die Theorie dahinter<br>- Anstatt eine strenge Trennung, wird es mit Hilfe einer Margin klassifiziert.<br>KONTRA:<br>- Keine Erfahrung in der Wahl der Kernel-Funktion: Können nicht volles Potential ausschöpfen<br>- Eher für höher-Dimensionale Daten |
| Decision Tree | PRO:<br>- Sehr gut bekannt und häufig positive Ergebnisse gesehen<br>- Uns bekannt und entsprechend wissen wir die Theorie dahinter (z.B. Anzahl der Blätter, Höhe des Baumes, etc.)<br>- Ebenfalls schneller Prozess für unsere Datenmenge<br>- Intuitiv<br>- Gut für kategorische Spalten<br>KONTRA: Uns ist kein Nachteil bewusst |
| Multi-Layer-Perceptron Classifier (Neuronale Netze) | PRO:<br>- Bekannt für gute Performance<br>- Für unsere Datenmenge relativ schnell<br>KONTRA:<br>- Zu wenig Samples (wird genauer bei vielen Samples)<br>- Komplizierte Einstellung von Hyperparametern<br>- Ggf. langer GridSearchCV() notwendig mit hoher Laufzeit |

In der unteren Tabelle sind die jeweiligen Algorithmen mit der Laufzeit und dem Train- und Testscore zu sehen.

- *_default* sind die Algorithmen ohne Parametereinstellungen
- *_adapted* sind diejenige gemeint, die wir händisch eingestellt haben (Hinweis: Wird später erklärt).
- *_grid* sind diejenige gemeint, die mit GridSearchCV den best estimator gab (Hinweis: Wird später erklärt).
- *Laufzeit -1:* Hier gibt es keine Laufzeit, denn sie wurde nicht gemessen.

Die Ausgabe der beiden „Scores“ dient zur Einschätzung von möglichem Overfitting der Modelle.

| | Model | Train-Score | Test-Score | Laufzeit |
|---|---|---|---|---|
| 0 | bayes_default | 0.767932 | 0.797753 | 0.002s |
| 1 | logreg_default | 0.798875 | 0.803371 | 0.031s |
| 2 | knn_default | 0.856540 | 0.808989 | 0.065s |
| 3 | knn_10_n | 0.831224 | 0.797753 | 0.114s |
| 4 | svc_default | 0.815752 | 0.797753 | 0.038s |
| 5 | svc_c10 | 0.834037 | 0.808989 | 0.111s |
| 6 | svc_grid | 0.834037 | 0.808989 | -1 |
| 7 | tree_default | 0.985935 | 0.769663 | 0.003s |
| 8 | tree_adapted | 0.890295 | 0.837079 | 0.003s |
| 9 | tree_grid | 0.881857 | 0.780899 | -1 |
| 10 | mpl_default | 0.825598 | 0.820225 | 0.969s |
| 11 | mpl_adapted | 0.821378 | 0.831461 | 1.158s |
| 12 | mlp_grid | 0.814346 | 0.803371 | -1 |
| 13 | forest_adapted | 0.894515 | 0.820225 | 0.237s |
| 14 | forest_grid | 0.985935 | 0.792135 | -1 |

*Abbildung 1: Modell-Übersicht*

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann

Wie bereits in der Tabelle mit den Vorüberlegungen zu sehen, ist die Laufzeit bei allen Modellen sehr schnell, da unser Datenset vergleichsweise klein ist.

## Maschine-Learning Modelle optimieren:

Aufgrund der primären Scores erschienen uns die folgenden Algorithmen bezüglich einer Optimierung sinnvoll:

KNN, SVC, DECISION TREE und MLP

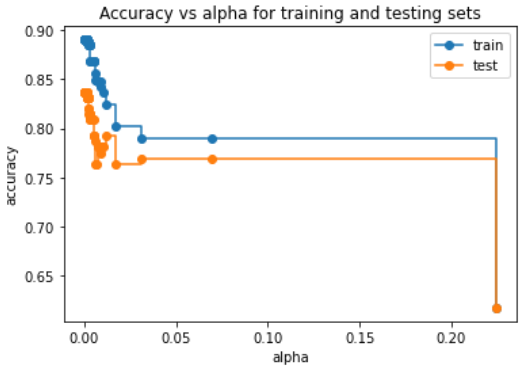
Eine zunächst händische Optimierung wurde für die folgenden Parameter mit den folgenden Begründungen durchgeführt. (Vergleiche für die entsprechenden Outcomes bezüglich der Scores auch die *Abbildung 1: Modell-Übersicht*).

| Algorithmus | Angepasste Hyperparameter |
|---|---|
| Kneighbors Classifier | n_neighbors: *Wie viele Nachbarn werden zur Klassifizierung genutzt?*<br>Werte von 2 – 10 genutzt, da sinnvolle Anzahl für Nachbarn bezüglich der Samplegröße- default-Wert 5 am besten<br><br>metric: *Wie werden die Abstände gemessen?*<br>Sehr wichtige Funktion. Da aber Verteilung im Raum der Daten nicht 100% klar ist, wurde diese im Defaultmodus belassen. Hohes Potential für weitere Verbesserung in die Zukunft. |
| Support Vector Classification | C: *Wie streng ist die Regularisierung? (Desto größer, desto schwächer)*<br>Werte von 0.001 – 10 genutzt. Eine höhere Zahl (also eine schwächere Reg.) war nicht sinnvoll. C = 1 im default hat sich am besten erwiesen. Auch eine geringere Regularisierung erbringt keine Verbesserung.<br><br>kernel: *Durch welche Formel werden die Daten getrennt?*<br>Unwissen über Kernel-Funktion und welche für Datenverteilung geeignet. Hohes Potential für weitere Verbesserung in die Zukunft.<br><br>degree: *Welchen Polynomgrad soll kernel haben?*<br>Nur dann wichtig, wenn z.B. „poly“ bei kernel ausgewählt wurde; nach händischen Versuchen verworfen.<br><br>decision_function_shape: *Welche Gruppen (Eins gegen Eins bzw. Eine gegen alle restlichen) werden verglichen?*<br>Aufgrund von bisherigen Erfahrungen kein großer Unterschied zu erwarten. Letztendlich auch kaum eine Veränderung der Scores. Letztendlich doch für ovr (default) entschieden. |
| Decision Tree | Criterion: *Welches Qualitätsmaß soll angewandt werden?*<br>Gini spart Rechenleistung im Vergleich zur Entropy. Entropy ist aber genauer. Bei unserem kleinen Datenset ist der Vorteil der Rechenleistung jedoch zu vernachlässigen.<br><br>max_depth: *Wie „tief“ also wie viele Ebenen soll der Baum haben bis zum Ende der Klassifizierung?*<br>Hat sich händisch nicht als sehr sinnvoll erwiesen. Hier lieber über die Anzahl der leafs gearbeitet. |

**min_samples_leaf:** *Wie viele Samples sind zum Split notwendig?*
Nach händischem Ausprobieren im Bereich von 1 bis 50 kam der beste Score hier für 4 Blätter heraus.

**gamma:** *Unklar (später im GridSearchCV).*

**ccp_alpha:** *Was ist der Grenzwert für die Komplexität des Baumes? (Beschneiden des Baumes, wenn zu Komplex)*
Unsicherheit bezüglich der Bedeutung des Parameters. Darstellung durchgeführt, ob eine Anpassung des Parameters helfen könnte bezüglich overfittings (siehe Abbildung lunten „Accuracy vs alpha for training and testing sets“). Aufgrund der geringen zu erwartende Effekte Anpassung verworfen.




Multi-Layer-Perceptron Classifier

**activation:** *Welche Funktion soll zur Datentrennung genutzt werden in jedem Knotenpunkt?*
Unterschiedliche Aktivierungsfunktionen führen zur unterschiedlichen Trennung der Daten. Relu ist default und wurde von uns genutzt, denn diese kann sich am besten um Daten „schlängeln“. Entsprechend im Händischen so gelassen. **Hohes Potential für weitere Verbesserung in die Zukunft.**

**hidden_layer_sizes:** *Wie viele Layers mit wie vielen Knoten?*
Hier herausgefunden, dass die Tupel Form bestimmt, wie viele Hidden-Layers es gibt (z.B. (30,20) entspricht 2 Layern mit 30 Knoten im ersten und 20 im zweiten).
Aufgrund der Unerfahrenheit und da es sich um einen kleinen Datensatz handelt, wird davon ausgegangen, dass ein Layer ausreicht. Händischer Versuch mit Zahlen zwischen 200 und 3 Knoten. Bestes Outcome bei 30 Knoten. **Hohes Potential für weitere Verbesserung in die Zukunft.**

**solver:** *Wie werden die Gewichte optimiert?*
Händisch versucht, aber letztendlich für adam entschieden, da besten Werte. Obwohl in Doku steht, dass adam für große Datensätze angewendet werden sollte.

**learning_rate/ learning_rate_init:** *Wie groß sind die „Schritte“ des Algorithmus hin zum Minimum?*
Hier für kleine Rate entschieden (default), da Sample klein. Mit größeren Schritten bräuchte man weniger Iterationen, aber für unser Datenset unproblematisch

**max_iter:** *Wie viele Iterationsdurchgänge?*

Bei uns erhöht, da es sonst zu keiner Konvergenz kam. Möglicherweise aufgrund der kleinen learning rate. Aber für unseren Datensatz unproblematisch

Überlegungen zu diesem Schritt auf unterschiedlicher Weise zustande:

- Diskussionen und intensive Beschäftigung mit den Parametern und der Dokumentation des sklearn Modells. Weiterhin wurde in Foren von ResearchGate und ähnliche Foren durchgelesen und anschließend mit der Gruppe besprochen.
- Gute Ergebnisse durch einfaches Nachdenken, was sinnvolle Bereiche sein könnten.
- Nicht immer gibt es eine Besserung des Scores durch die Angabe von eigenen Parametern. Entsprechend wurde bei KNN erkannt, dass mit dem Default-Werte bessere Scores kamen.

## Maschine-Learning Modelle optimieren mit GridSearchCV:

Für die vier ausgewählten Algorithmen wurden mehrere GridSearchCV zur automatischen Parameteroptimierung durchgeführt.

```
model = GridSearchCV(estimator, parameter, scoring='neg_mean_squared_error', n_jobs=-1, cv=5)
model.fit(self.xTrain, self.yTrain)
```

Außerdem wurden alle Cores für des Computer für eine schnellere Verarbeitung genutzt (n_jobs = -1). Die Cross-Validation (cv = 5) wurde ebenfalls zur schnelleren Verarbeitung, niedrig gehalten.

Bei den GridsearchCV wurden einige der o.g. Parameter, welche sich schon durch die händische Anpassung als „fruchtbar“ erwiesen, in entsprechend sinnvollen Grenzen, durchgeführt.

Es wurden außerdem Hyperparameter einbezogen, die uns unklar blieben aber möglicherweise einen Einfluss auf das Outcome haben könnten. (z.B. *solver* bei MLP, *gamma* bei SVC). Falls diese sich als gute Hyperparameter erwiesen hätten, wäre eine nähere Betrachtung in sinnvoll gewesen.

Bei allen GridSearchCV ergaben sich ein durchgehend schlechteres Outcome als bei den händisch gewählten Parametern (Vergleiche Abbildung 1: Modell-Übersicht). Als positiv ist zu bewerten, dass unser GridSearchCV aufgrund der gut gewählten Parameter und des kleinen Samples vergleichsweise schnell verlief.

Warum der GridSearchCV auch mit Parametern, die sich schon in der händischen Auswahl als sinnvoll erwiesen, keine besseren oder zumindest gleiche Ergebnissen liefert blieb, auch nach mehrfachem Recherchieren und Debuggen, unklar. Eventuell könnte man hier das RandomizedSearchCV einsetzen und man hätte bessere Ergebnisse erhalten, der jedoch aus Zeitgründen nicht durchgeführt wurde.

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann

## Ensemble Methoden – Voting Classifier

Als zusätzliche Ensemble-Methode entschieden wir uns für einen Voting-Classfier, der für uns verständlicher war mit der soft-Methode, die die prozentuale Anteile betrachtete und anschließend die Klassifizierung zuordnet.

```
voting_clf = VotingClassifier(
    estimators=[('DecisionTree', TREE_model), ('SVC', SVC_model), ('KNN', knn_model)],
    voting='soft')      # soft (prozentuale Anteil wird berechnet und anschließend zugeordnet)

for clf in (TREE_model, SVC_model, knn_model, MLP_model, voting_clf):
    clf.fit(xTrain, yTrain)     # fitten jeweils der oben bessere Modelle
    yPred = clf.predict(xTest)      # Predicten mit der xTest-Menge
    print(clf.__class__.__name__, accuracy_score(yTest, yPred))    # printen des Scores zu jedem Algorithmus
```

Zunächst habe wir alle 4 Klassifizierungsalgorithmen, welche sich in den vorherigen Schritten als leistungsstark erwiesen haben, für das Voting einbezogen. Hierbei zeigte sich jedoch, dass sich der Voting-Classifer an dem MLP-Classifer orientierte und die entsprechenden Scores hier 100% übereinstimmten. Aufgrund dieses Verhaltens, der erwünschten ungeraden Anzahl der im Voting-Classifer genutzten Sub-Classifier, und der insgesamt schlechteren Verhaltensweise des MLP-Classifiers im Vergleich zum Decision-Tree – Classifiers habe wir uns letztendlich dafür entschieden, den MLPClassifier auszuschließend und nur drei Klassifizierungsalgorithmen zu nehmen.

Bei dem Vergleich der Scores haben wir hier ein schlechteres Ergebnis im Vergleich zu dem adaptierten DecisionTreeClassifier. Entsprechend wollten wir mit dem DecisionTreeClassifier weitermachen und schauen, ob mit dem RandomForestClassifier wir bessere Ergebnisse erzielen können.

Wir wählten die gleichen Hyperparameter wie beim adapted DecisionTreeClassifier und haben auch hier ein schlechteren Score erhalten. Durch den GridSearchCV wurde dies auch nicht besser, sodass wir letztlich für die Predicition der Daten von den ursprüngliche test.csv (ohne die Labels) mit dem adapted DecisionTreeClassifier durchführen. Die predicted Labels können jetzt auf Kaggle hochgeladen werden und mit dem richtigen Daten verglichen werden.

## Fazit

Das zu Beginn festgelegte Ziel war:

> Aus gegebenen Informationen zu Passagieren der Titanic Vorhersagen zu deren Überleben treffen.

Dazu haben wir:

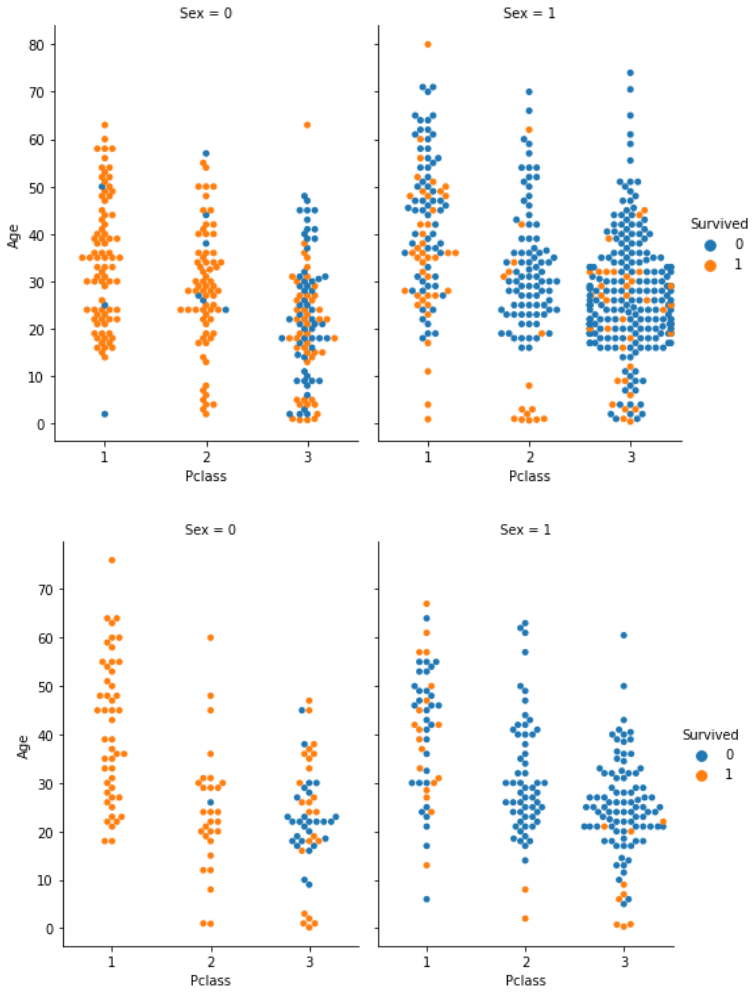
- den Datensatz vorverarbeitet;
- verschiedene Methoden des Machinellen Lernens angewendet;
- Parameter manuell und automatisiert optimiert;
- Eine Ensemble Methode verwendet.

Auf den von uns abgeteilten Testdaten haben wir eine Genauigkeit von bis zu 83% erreicht.

Dabei war die manuelle Optimierung der Parameter durchweg der automatisierten Optimierung überlegen. Dies verwundert uns und ließ uns desillusioniert zurück.

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann

Die nächsten beiden Abbildungen zeigen die tatsächlichen Label (Survived ja/nein) für den kaggle-train-Datensatz (oben) sowie die von uns vorhergesagten Label (Survived ja/nein) für den kaggle-test-Datensatz (unten) jeweils bezüglich der Features Sex, Age, Pclass. Die beiden Abbildungen zeigen eine gute visuelle Übereinstimmung.




Auffällig ist, dass unsere Vorhersage keine überlebenden für Sex=male, Pclass=2 oder 3 und Age>25 für Pclass 3 bzw. Age > 15 für Pclass 2 (unteres Bild rechts außen) vorhersagt. Dies ist ein Unterschied zu den bekannten Labeln (oberes Bild rechts außen) und könnte auf eine Überregularisierung hindeuten.

Da uns eine 80% Genauigkeit noch nicht ausreicht, schlagen wir folgende Verbesserungen vor:

- Informationen aus Spalte „Cabin“ doch verwenden, um Lage der Kabine bzgl. Rettungsboote festzustellen.
- Alter gruppieren, um weniger kontinuierliche Daten zu haben und dann die Algorithmen entsprechend tunen.
- RandomizedSearchCV als Vorstufe zu GridSearchCV
- Mehr Wissen über die Algorithmen zur besseren Parameterwahl
- Gibt es einen Zusammenhang Hafen und Klasse? „Reiche-Leute-wohnen-alle-zusammen“
- Andere Algorithmen nutzen

Gruppe E: Helena Brinkmann, Esra Lenz, Abdurrahman Derin, Esther Klann