Regressionsverfahren

- Phase: Data Modelling -

Ziel Prognose der Ausleihvorgänge insgesamt für jeweils eine Stunde eines Tages mithilfe eines geeigneten Machine Learning Modells!

Zielvariable: Ausleihvorgänge (count_out)

Dieses Notebook nutzt die Dateien counts_prepared.pkl. In dem vorliegenden Notebook werden keine Dateien erzeugt.

In [31]:

```
from sklearn import linear_model
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import statsmodels.api as sm
import matplotlib as plt
import seaborn as sns
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
```

In [32]:

```
DATA_PATH = '../data/'
```

In [33]:

```
df_count_weather = pd.read_pickle(DATA_PATH+'counts_prepared.pkl')
df_count=pd.read_pickle(DATA_PATH+'counts_prepared.pkl')

df_count.reset_index()
df_count['count_out'].sum()
df_count = df_count.drop(['count_in'], axis=1)
df_count.reset_index(inplace=True)
```

Erstellen von Trainings- und Testdaten

Trainingsdaten mit den Datensätzen von 2015 und 2016 werden erstellt Testdaten werden mit dem Datensatz von 2017 erstellt.

Entnommen aus dem ***70_Data Preperation*** Notebook.

In [34]:

```
array = [2015, 2016]
```

In [35]:

```
dataTrain = df_count.loc[df_count['year'].isin(array)]
dataTrain = dataTrain.sort_values(by=['date'])
dataTest = df_count.loc[df_count['year']==2017]
dataTest = dataTest.sort_values(by=['date'])

datetimecol = dataTest['date']
yLabels = df_count["count_out"]
```

In [36]:

```
# Trennen von Trainings- und Testdaten in unabhängige Variablen (X) und abhängige Varia
ble (count_out y)
drop_cols = ['date', 'count_in', 'index']
y_cols =['count_out']
feature_cols = [col for col in df_count.columns if (col not in y_cols) & (col not in dr
op_cols)]

X_train = dataTrain[feature_cols]
X_test = dataTest[feature_cols]

y_train = dataTrain['count_out']
y_test = dataTest['count_out']
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(17401, 41) (17401,) (8560, 41) (8560,)
```

Feature Scaling

Durch das Feature Scaling sollen die Daten so transformiert werden, dass deren Mittelwert der Verteilung bei 0 liegt und deren Standardabweichung bei 1. Dadurch soll sichergestellt werden, dass die Daten Standardnormalverteilt sind, was eine Voraussetzung für die Lineare Regression ist.

Quelle: Schritt 6: https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html (https://www.kdnuggets.com/2019/07/data-pre-processing-optimizing-regression-model-performance.html) https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

In [38]:

```
# Vor einer Regression sollten die Werte skaliert werden --> durch StandardScaler werde
n die Werte standaridsiert
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Definition der Kostenfunktion

Diese Funktion wird später verwendet um die Messwerte RMSLE und R-Squared Value zu berechnen, um die Güte der Prognose zu bewerten.

In [39]:

```
# Implementierung des RMSLE
def rmsle(y_test, y_pred):
    rmsle = np.sqrt(metrics.mean_squared_log_error( y_test, y_pred ))
    #msle = metrics.mean_squared_log_error(y_test, y_pred)
    #rmse = math.sqrt(mse)
    return (rmsle)
```

In [40]:

```
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
def score_model(model):
    """
    Fits a model using the training set, predicts using the test set, and then calculat
es
    and reports goodness of fit metrics.
    """
    model.fit(X_train, y_train)
    yhat = model.predict(X_test)
    r2 = r2_score(y_test, yhat)
    me = rmsle(y_test, yhat)
    print("Results from {}: \nr2={:0.3f} \nRMSLE={:0.3f}".format(model, r2,  me))
```

Regressionsmodell erstellen

Da bei der Aufteilung der Trainings- und Testdaten durch "feature_cols" alle Spalten außer "count_out", "date", "count_in" und "Index" betrachtet werden, wird keine nachfolgend keine reine lineare Regression erstellt sondern eine Multiple Lineare Regression.

Es werden ebenso zwei Varianten darstellt, wie eine Lineare Regression in Python erstellt werden kann. Nachfolgend wird zum Vergleich noch eine einfache lineare Regression erstellt, welche als Unabhängige Variable X nur die Temperatur und nicht mehr alle Spalten verwendet.

Variante 1: Multiple Lineare Regression mit scikit-learn erstellen

In [41]:

```
X = df_count_weather[feature_cols]
y = df_count_weather.count_out

#Instanziieren der Klasse
lm = LinearRegression()

#Modell trainieren
lm.fit(X_train,y_train)

#Vorhersage
y_pred =lm.predict(X_test)

#Metriken ausgeben
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error (RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_p
red)))

#R-Squared Wert berechnen
print('R-Squared Value:', lm.score(X_test, y_test))

#RMSLE berechen:

#score_model(lm)
#from sklearn.metrics import mean_squared_log_error
#print(rmsle = np.sqrt(metrics.mean_squared_log_error( y_test, y_pred )))

#Urspürngliche Berechnung mit der oben definierten Kostenfunktion für RMSLE funktionier
te nicht.
#Daher Berechnung wurde die Berechnung nach dem Code von https://towardsdatascience.co
m/metrics-and-python-850b60710e0c ausprobiert.

import math
def RMSLE(y_pred, y_test):
    total = 0
    for k in range(len(y_pred)):
        LPred= np.log1p(y_pred[k]+1)
        LTarg = np.log1p(y_test[k] + 1)
        if not (math.isnan(LPred)) and  not (math.isnan(LTarg)):
            total = total + ((LPred-LTarg) **2)

    total = total / len(y_pred)
    return np.sqrt(total)
print ('My RMSLE: ' + str(RMSLE(y_pred,y)) )
```

```
Mean Absolute Error: 157.5653218168725
Mean Squared Error: 48952.735047934984
Root Mean Squared Error (RMSE): 221.252649809974
R-Squared Value: 0.6420604920286261

C:\Users\tanja\anaconda3\lib\site-packages\ipykernel_launcher.py:33: Runti
meWarning: invalid value encountered in log1p

My RMSLE: 1.867908291805847
```

Der MSE (Mean Squared Error) Wert sollte ein relativ kleiner Wert sein. Der R-Squared Wert sollte ein Wert möglichst nahe an 1,0 sein. Ist der Wert sehr gering, heißt dass, das die unabhängigen X-Variablen nicht so gut dafür geeignet sind die abhängige y-Variable vorherzusagen. Ist der Wert nahe an 1 besitzt das Modell eine gute Anpassungsgüte. </>

In [42]:

```
#Accurancy berechnen
from sklearn.model_selection import cross_val_score
accuracy = cross_val_score(estimator = lm, X = X_train, y = y_train, cv =10)
accuracy.mean()
```

Out[42]:

```
0.5287391446868062
```

Dieses Prognosemodell hat eine Vorhersagegenauigkeit von ca. 52%.

In [45]:

```
#Für ein paar Werte die Unterschiede zwischen Vorgehergesagt und Testdaten anschauen:
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df1 = df.head(25)
df1.head(10)
```

Out[45]:

	Actual	Predicted
17401	77	-67.745248
17424	46	23.843733
17423	60	122.434287
17422	91	211.978264
17421	110	315.290045
17420	162	487.082818
17419	184	710.825706
17417	390	586.986311
17416	546	450.887529
17415	522	375.716275

Es ist hierbei exemplarisch zu erkennen, dass das Prognosemodell in einigen Fällen mit dem Vorhergesagten Wert extrem neben dem tatsächlichen Wert daneben liegt.

Variante 2: Mulitple lineare Regression mit 'statsmodel' Package erstellen

Nachfolgend wird die multiple lineare Regression noch mit einer anderen Vorgehensweise erstellt.

In [16]:

```
## Create a fitted model
lm1=sm.OLS(y_train, X_train).fit()
predictions =lm1.predict(X_test)

#print summary
lm1.summary()
```

Out[16]:

OLS Regression Results

Dep. Variable:	count_out	**R-squared (uncentered):**	-32.506
Model:	OLS	**Adj. R-squared (uncentered):**	-32.585
Method:	Least Squares	**F-statistic:**	-410.8
Date:	Fri, 04 Sep 2020	**Prob (F-statistic):**	1.00
Time:	09:05:53	**Log-Likelihood:**	-1.2959e+05
No. Observations:	17401	**AIC:**	2.593e+05
Df Residuals:	17360	**BIC:**	2.596e+05
Df Model:	41		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-6.5077	3.155	-2.063	0.039	-12.692	-0.324
x2	99.3858	43.232	2.299	0.022	14.646	184.125
x3	-62.8514	19.098	-3.291	0.001	-100.286	-25.417
x4	-7.5484	3.261	-2.315	0.021	-13.939	-1.157
x5	7.8312	3.672	2.132	0.033	0.633	15.029
x6	-0.7805	38.010	-0.021	0.984	-75.284	73.723
x7	-16.9234	3.495	-4.842	0.000	-23.775	-10.072
x8	10.9524	3.269	3.350	0.001	4.545	17.360
x9	7.3744	3.168	2.328	0.020	1.165	13.584
x10	41.9585	5.068	8.279	0.000	32.025	51.892
x11	42.8546	4.683	9.150	0.000	33.675	52.034
x12	37.4378	6.680	5.605	0.000	24.345	50.531
x13	-9.5727	4.363	-2.194	0.028	-18.124	-1.021
x14	-14.1392	4.377	-3.231	0.001	-22.718	-5.560
x15	-16.2870	4.397	-3.704	0.000	-24.905	-7.669
x16	-17.4889	4.422	-3.955	0.000	-26.156	-8.821
x17	-16.6288	4.441	-3.744	0.000	-25.334	-7.924
x18	-6.7826	4.457	-1.522	0.128	-15.518	1.953
x19	19.5967	4.473	4.381	0.000	10.828	28.365
x20	83.2918	4.493	18.540	0.000	74.486	92.098
x21	136.1515	4.503	30.237	0.000	127.325	144.977
x22	74.4752	4.517	16.488	0.000	65.621	83.329
x23	57.3899	4.531	12.666	0.000	48.509	66.271
x24	71.5555	4.541	15.758	0.000	62.655	80.456
x25	87.5478	4.545	19.262	0.000	78.639	96.457
x26	86.8657	4.537	19.146	0.000	77.973	95.759
x27	82.1481	4.514	18.200	0.000	73.301	90.995

x28	88.8719	4.472	19.871	0.000	80.106	97.638
x29	114.8206	4.435	25.889	0.000	106.127	123.514
x30	148.9165	4.274	34.843	0.000	140.539	157.294
x31	134.3200	4.324	31.066	0.000	125.845	142.795
x32	88.6181	4.367	20.294	0.000	80.059	97.177
x33	53.8981	4.362	12.356	0.000	45.348	62.448
x34	31.4560	4.362	7.211	0.000	22.905	40.007
x35	15.6016	4.361	3.577	0.000	7.053	24.150
x36	17.5696	4.142	4.242	0.000	9.450	25.689
x37	7.8440	4.197	1.869	0.062	-0.383	16.071
x38	4.3529	4.131	1.054	0.292	-3.744	12.450
x39	11.6014	4.136	2.805	0.005	3.495	19.707
x40	10.0786	4.121	2.445	0.014	2.000	18.157
x41	15.7346	4.121	3.818	0.000	7.657	23.812

Omnibus:	593.168	**Durbin-Watson:**	0.238
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1434.562
Skew:	0.169	**Prob(JB):**	0.00
Kurtosis:	4.365	**Cond. No.**	31.9

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Die Spalte coeff gibt an, wie hoch der Zusammenhang der Variable mit der abhängigen y-Variablen (count_out) ist. Ein hoher Wert sagt aus, dass ein hoher Zusammenhang zwischen diesen beiden Variabeln besteht.

Eine weitere Verbesserung dieses Prognosemodells könnte durch eine Polynomregression statt durch eine Lineare Regression erzielt werden. Diese wurde aber nicht weiter ausgeführt, da im Vergleich mit den anderen Prognosemodellen, andere Modelle schon eine bessere Prognosegenauigkeit erreicht haben und daher der Fokus mehr auf die Verbesserung dieser Modelle gelegt wurde.

Lineare Regression zum Vergleich

Zum einfachen Vergleich wird nun noch eine reine lineare Regression erstellt. Als Unabhängige Variable X wird nur die Spalte Temperatur verwendet (abhängige Variable weiterhin"count_out"):

In [17]:

```
# Trennen von Trainings- und Testdaten in unabhängige Variablen (X) und abhängige Variable (count_out y)
# Nur die Temperaturspalte wird nicht bei Drop Coloum hinzugefügt.
drop_cols = ['date', 'count_in', 'index', 'day_of_week', 'dewpoint', 'humidity', 'precipitation', 'pressure','windspeed', 'workingday', 'year', 'Season_Fall', 'Season_Spring', 'Season_Summer', 'hour_0', 'hour_1','hour_2', 'hour_3', 'hour_4', 'hour_5', 'hour_6', 'hour_7', 'hour_8', 'hour_9', 'hour_10', 'hour_11', 'hour_12','hour_13', 'hour_14', 'hour_15', 'hour_16', 'hour_17', 'hour_18', 'hour_19', 'hour_20', 'hour_21', 'hour_22','weekday_Friday', 'weekday_Monday', 'weekday_Saturday', 'weekday_Thursday', 'weekday_Tuesday', 'weekday_Wednesday']
y_cols =['count_out']
feature_cols = [col for col in df_count.columns if (col not in y_cols) & (col not in drop_cols)]

X_train = dataTrain[feature_cols]
X_test = dataTest[feature_cols]

y_train = dataTrain['count_out']
y_test = dataTest['count_out']
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)


#Instanziieren der Klasse
lm = LinearRegression()

#Modell trainieren
lm.fit(X_train,y_train)

#Vorhersage
y_pred =lm.predict(X_test)

#Metriken ausgeben
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error (RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

#R-Squared Wert berechnen
#print('R-Squared Value:', lm.score(X_test, y_test))
score_model(lm)
```

```
(17401, 1) (17401,) (8560, 1) (8560,)
Mean Absolute Error: 292.47451897229865
Mean Squared Error: 132327.0645464576
Root Mean Squared Error (RMSE): 363.7678717897687
Results from LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False):
r2=0.032
RMSLE=1.644
```

Man kann bei diesem Test sehen, dass die Lineare Regression allein mit der Temperatur im Verhältnis zu dem count_out weit aus schlechter performt, als die vorherige Multiple Lineare Regression bei der alle anderen Spalten auch mitbeachtet werden.

Eine solche Lineare Regression könnte genutzt werden, wenn nur gewisse Hypthosen überprüft werden sollen z.B. welchen Einfluss die Temperatur auf die Ausleihvorgänge hat im Vergleich dazu welchen Einfluss eine andere Spalte auf die Ausleihvorgänge hat. Dann könnte für beide Fälle eine Lineare Regression erstellen und die Güte und die Koeffizienten miteinander vergleichen.