Aufgabe3_test

November 21, 2020

1 Aufgabe 3: Data Preparation und Modeling

1.1 TODO:

- Anomalien bereinigen
- Feature Engineering
  - Unnötige Spalten (siehe Aufg. 2) entfernen
  - One-Hot Encoding einiger Eigenschaften
-

[11]:
```
import pandas as pd
import matplotlib.pyplot as plt
import math
import seaborn as sb
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.model_selection import train_test_split
```

[3]:
```
dataset = pd.read_csv('./DatenAusgegeben1.0.csv', delimiter=';', encoding =␣
↪'cp852')
dataset
```

[3]:

	Grundstück in qm	Grundstücksform	Steigung	Bezirk	Zone	Lage	\
0	898	IR1	Nein	Somerset	RL	Norm	
1	1326	Reg	Nein	North East	RL	Norm	
2	725	Reg	Nein	Somerset	RL	Norm	
3	725	Reg	Nein	Somerset West	RL	Norm	
4	697	Reg	Nein	Miller	RL	Norm	
…	…	…	…	…	…	…	
1995	1272	IR1	Nein	North East	RL	Norm	
1996	941	IR1	Nein	Miller	RL	Norm	
1997	1093	Reg	Nein	Grand Park	RL	Norm	
1998	1228	IR1	Nein	North East	RL	Norm	
1999	778	Reg	Nein	Somerset	RL	Norm	

	Typ	Zustand	Gebaut	Renoviert	…	Schlafzimmer	Küchen	\
0	2Fam	4	2107	2107	…	6	2	
1	1Fam	5	2133	2133	…	3	1	

```
2     1Fam         7    2096       2138  …              3       1
3     1Fam         5    2135       2135  …              3       1
4     1Fam         5    2129       2129  …              3       1
…     …       …      …          …  …             …      …
1995  1Fam         5    2133       2134  …              3       1
1996  1Fam         5    2134       2134  …              3       1
1997  1Fam         7    2040       2130  …              2       1
1998  1Fam         5    2134       2134  …              3       1
1999  1Fam         7    2071       2103  …              2       1

     Küchenqualitt Rume   Garage Typ  Garagenkapazitt  Pool  Verkaufsmonat  \
0                3   10        Anbau                2   NaN              8
1                4    7        Anbau                3   NaN              1
2                4    5  Freistehend                1   NaN              6
3                4    8        Anbau                2   NaN              4
4                4    7        Anbau                2   NaN              6
…              …  …          …                …   …           …
1995             5   10    Eingebaut                3   NaN              3
1996             4    7        Anbau                2   NaN              7
1997             3    6  Freistehend                1   NaN              5
1998             4    9    Eingebaut                3   NaN              2
1999             4    6       2Typen                3   NaN              6

      Verkaufsjahr   Preis
0             2136  156500
1             2137  350000
2             2140  137900
3             2136  184000
4             2140  189000
…             …     …
1995          2138  412500
1996          2139  195500
1997          2137  112000
1998          2136  279000
1999          2138  152400

[2000 rows x 28 columns]
```

1.2 Unnötige Spalten entfernen

Zu entfernen sind:
Siehe Aufg. 2. Untersuchung auf einen Zusammenhang zwischen Preis und Verkaufszeitpunkt: - Verkaufsjahr - Verkaufsmonat

(Da Gesamtwohnfläche zusammengesetzt aus erster Stock und zweiter Stock ist: - Erster Stock in qm - Zweiter Stock in qm #TODO: Prüfen ob nicht rauszunehmen)

Siehe Aufg. 2. Untersuchung des Datensatzes auf vernachlässigbare Hauseigenschaften (Spalten):

- Pool - Küchen - Klimaanlage - Heizung

[4]:
```
df_keinunnoetig = dataset.drop(['Verkaufsjahr', 'Verkaufsmonat', 'Erster Stock␣
↪in qm', 'Zweiter Stock in qm', 'Pool', 'Küchen', 'Klimaanlage', 'Heizung'],␣
↪axis = 1)
```

1.3 Anomalien bereinigen

Nun sollen Anomalien in dem Datensatz gefunden und bereinigt werden. Hierbei wird der Interquartilsabstand verwendet um Ausreißer zu finden.
(https://en.wikipedia.org/wiki/Interquartile_range)

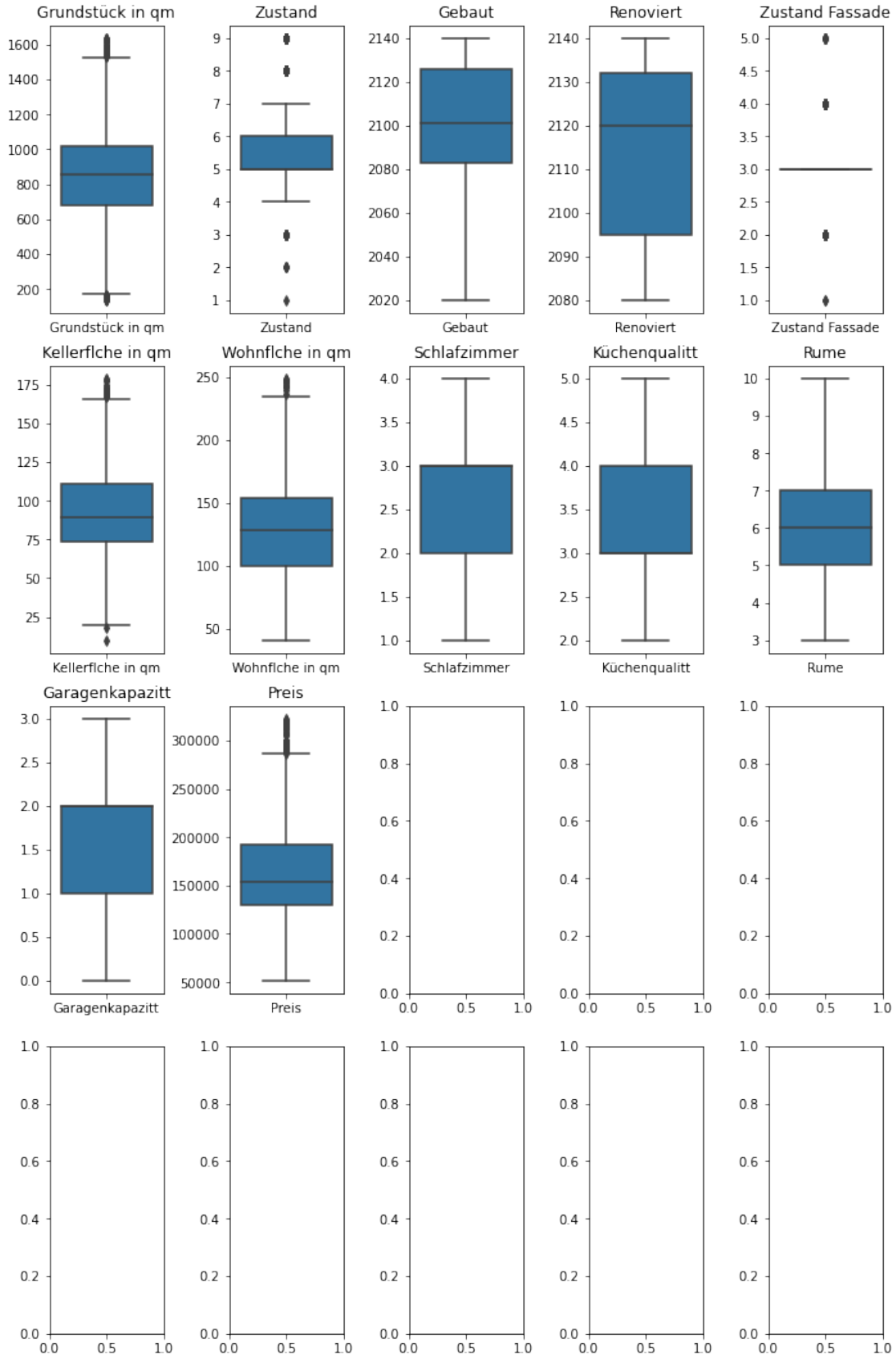
Hierbei sollen ausschließlich numerische Werte, also keine Werte einer ordinalen Skala betrachtet werden. (`columns_filtered`)

[5]:
```
columns = ['Grundstück in qm', 'Zustand', 'Gebaut',
           'Renoviert', 'Zustand Fassade', 'Kellerflche in qm',
           'Wohnflche in qm', 'Schlafzimmer', 'Küchenqualitt',
           'Rume', 'Garagenkapazitt', 'Preis']

columns_filtered = ['Grundstück in qm', 'Kellerflche in qm', 'Wohnflche in qm',
           'Schlafzimmer', 'Gebaut', 'Rume', 'Garagenkapazitt', 'Preis']

Q1 = df_keinunnoetig[columns_filtered].quantile(0.25)
Q3 = df_keinunnoetig[columns_filtered].quantile(0.75)
IQR = Q3 - Q1

df_anomalienbereinigen = df_keinunnoetig[~((df_keinunnoetig[columns_filtered] <␣
↪(Q1 - 1.5 * IQR)) |(dataset[columns_filtered] > (Q3 + 1.5 * IQR))).
↪any(axis=1)]
```

1.4 Verifikation der Bereinigung

Nun soll verifiziert werden, ob die Bereinigung auf Anomalien erfolgreich gewesen ist. Hierzu können die untenstehenden Boxplots mit denen aus **Aufg. 2, Finden von Anomalien** verglichen werden.

[6]:
```
fig, axes = plt.subplots(4, 5, figsize = (10, 15))
i = 0
plt.tight_layout(pad = 2)
for column in columns:
  chosenax = axes[math.floor(i / 5)][i % 5]
  chosenax.set_title(column)
  sb.boxplot(data = df_anomalienbereinigen[[column]], ax = chosenax)
  i = i + 1
plt.show()
```


Grundstück in qm
Zustand
Gebaut
Renoviert
Zustand Fassade
Kellerflche in qm
Wohnflche in qm
Schlafzimmer
Küchenqualitt
Rume
Garagenkapazitt
Preis

[7]:
```
encode_kategorien = ['Bezirk', 'Lage', 'Heizungsqualitt', 'Typ', 'Steigung',␣
↪'Grundstücksform', 'Garage Typ', 'Zone']

encoded_kategorieeigenschaften = pd.get_dummies(dataset[encode_kategorien].
↪fillna('Keine'))
df_encoded = df_anomalienbereinigen.drop(encode_kategorien, axis = 1).
↪join(encoded_kategorieeigenschaften)
```

[41]:
```
import plotly.express as px
fig = px.parallel_coordinates(df_encoded, color = 'Preis', width = 1300, height␣
↪= 800)
fig.show()
```

[35]:
```
reg = LinearRegression()
reg2 = LogisticRegression()

df_train = df_encoded.drop('Preis', axis = 1)
values_predict = df_encoded['Preis']
x_train, x_test, y_train, y_test = train_test_split(df_train, values_predict,␣
↪test_size = 0.2)

reg.fit(x_train, y_train)
reg2.fit(x_train, y_train)
y_pred = reg.predict(x_test)

print(y_test.values, y_pred)

for bla in y_pred:
    print(type(bla))

# print(reg.coef_, reg.intercept_, reg.rank_)
print("Linear Regression", reg.score(x_test, y_test))
print("Logistic Regression", reg2.score(x_test, y_test))

from sklearn.metrics import classification_report
classification_report(y_test.values, y_pred)
```

```
[193500 275000 119900 149300 254750 159500 155000 274300 188500 230000
 147000 135000 137000 130000 135000 220000 215000 204000 134900 312500
 194000  80000 159000 168000  72000 162900 145250 257500 133700  99500
 233500 184100 306000 119500 180000  64000 179200 182000 235000 202000
 139000 218000 194700 157000  86000 214000 133000 175000 147000 246900
 177625 102000 180500 147000 163000 220000 290000 215000 240000 142500
 133000 184500 121000 122000 165000 215000  91000 189900  89000 128000
 165500 118900 159900 185000 191000 105000 240000  91300 214000 197500
```

```
 128000 174900 144900 116000 154000 114504 148000 185000 224000  96900
 109900 125000 145000 154000 146000 236000 202500 144100 285000 128000
 134450  85000 250000 110000 133000 167500 115000 184500 162500 174500
 145000 135500 156000 191500 189000 112900 172500 132000 127500 105000
 130000 200000 140000 128500 166000 155000 192500 147000 178000  85000
 136000 248000 142250 128000  99500 135500 137500 130000 220000 166000
 115000 150000 148500 116900 110000 226000 155000 289000 181000 246000
 148000 212500 179600 193000 111500 136500 187500 260000 144000 265000
 217000 275000 215000 159000 187000 111000 148000  89000 164000 123000
 124000 189000 129900 201000 164500  85500 260000 212000 313000  94900
 212300 184000 281213 123900 100500 140000 197000 159000 177000 247000
 143750 120500 181500 137000 119900 123500 127000 193000 195500 256000
 119164 160000 178000 143500 128000 149000 263550 122600 120875 168675
 144000 105000 226500 157500  91500 319900 145400 139000 129000 153900
 111750  83500  88250 275000 181000 149500 205000 142500 174000 125000
 224900 120500 122900 174000 145000 275000 186500 135000 137500 120000
 274000 280000 146300 159500 176000 109000 203000 219000 200000 167000
 120000 130000 140000 228500 184000 190000 157000 260000 159000 125500
 217500 315500 214000 135000 154900 124500 151500 138500 171500 113000
 306000 128500 309000 159000 187000 177000 133500 284000 153000 290000
 260000 138500 168500 239000 122500 178000 226001 169000 134000 114500
  79000 196000 178000 241500 173000 120000 255000 133500 213250 131000
 246990 175000 140000 110000 185000 181316 133000 236500 103500 117600
 228000 160000 155000 150000 162000 126000 165000 125000 147500 275500
 183200 128600 118000 139900 148500 124400 170000 135000 200000 159000
 137500 176000  87000 322500  93900 135000  94500 142000 172000 139000
 282000 144750 158450] [197481.17166009 277429.34947723 114538.11968616
147194.35819712
 236998.16610319 199844.41082248 187453.78770834 231457.94627184
 183244.41434726 216558.89200752 150920.98556941 152832.7344093
 131600.81257133 131553.75853259 153593.7076987  216827.29822462
 215124.34106461 206675.83916415 150046.43649016 265798.37949657
 246423.77585956  98067.76584564 163784.76732052 162172.11865886
  63728.69026107 162376.07579318 153721.77435961 263089.73136987
 117547.17602749  94878.60552532 231685.42076467 167971.45103828
 253834.72984271 122318.33147856 178493.49547329  63373.96131816
 197815.98992138 190014.40124363 206164.82190332 188836.2919956
 127585.70923119 187967.88590605 222514.59461686 157150.09652676
  99458.77942753 218814.57567832 132024.77657662 186705.93481582
 156713.57177182 219653.43456259 178372.78486256 104261.05684428
 205601.43987173 150173.49439175 152757.69537151 226757.49089572
 286784.90583682 204265.98907139 247094.44863662 128156.12991127
 137927.70401248 192613.85673791 119452.55250998 147065.70912452
 174149.1009697  203702.70703956  80783.49226162 189606.52629565
  81013.43367846 144919.70901431 171311.44154382 100469.10812148
 180796.15623196 186745.84526315 206982.52726816 114517.91325504
 229184.41487747  87958.7447371  251471.23546964 182102.3975388
 117827.18545025 156928.72328895 149055.50346254 119018.68879428
```

153409.86613157 108248.04622973 149386.12749938 189984.72377053
237906.14641502 100667.29606164 141006.59165888 120368.97988637
138032.95128201 158447.70177784 120139.73330948 226702.06151186
181117.86690364 152322.71682658 293826.60259328 161027.56843695
156348.97197993 76640.76008479 259502.80117226 110087.29945803
101783.71520202 142851.08332305 149047.19577925 202614.31651364
209688.09244076 178231.66707598 131975.59471799 116682.17530996
155031.97969369 207574.65031114 164091.72958988 108981.46820237
174564.79753536 138303.66909616 124809.28850882 99852.69329525
146034.57242455 201815.78348087 169654.19148597 133614.78327437
165387.48793738 138867.19413356 217543.76652647 125033.83814168
182671.21388952 87422.02449011 128448.95937624 272895.70460158
151071.97952705 122177.96504636 101753.19941095 156825.34952525
144239.10740325 130197.87475166 212155.15188937 114607.37242404
119445.63204343 213935.16436419 131948.0056254 100478.72308904
163802.30828976 211231.02005919 163503.64013036 264275.52269934
196775.9956836 228964.09651963 157891.71202588 226890.87243969
202895.01447078 208062.39361001 121689.64222057 123071.59866948
226405.11642255 301012.56377881 137737.12430939 287821.57272034
215421.24257189 216323.31466962 214290.8003199 154821.72351982
192138.90368663 104558.4055098 139059.0073917   90648.86357668
168815.69908318 128155.89230163 154515.33856993 184083.91969314
101085.31854185 210859.6053108 156883.64932898 91896.84735397
251367.22442391 211788.81469437 267230.10832553 93201.56072305
201495.20138004 175773.92415989 278738.71142839 119041.0833366
 98873.60651236 147145.04120684 207899.33817868 162793.04846587
177623.19968387 225853.25994563 147145.04120684 98548.91647465
213641.20588679 161449.06571595 108590.66820241 128926.14442532
126864.2646502 200361.45819066 195916.17247951 225613.44962176
154364.71126257 171480.54190404 169463.04755707 146841.3164845
141855.2945555 147212.46111231 269174.72861697 138406.83368857
138138.12246287 182820.2540425 147203.29143345 103711.78544683
211538.99548596 146125.87018972 92911.41866534 262262.41880454
135884.97637207 137306.97256802 125852.00386091 170730.33850898
109018.15531391 98508.12863255 86161.07200005 236754.34351166
200466.16840879 162953.54368562 194226.03681267 140432.3389611
190037.74138033 100991.1389332 224828.69946692 121916.9637868
113892.80649331 173466.42652367 122076.57236138 252587.63165032
206387.48245163 134183.1514527 124203.05729107 121766.88629635
243952.80417081 280310.90971094 142616.04183216 157263.27299872
179516.40592711 111551.69095795 207921.85370082 214365.31740588
199361.25267034 182126.88000754 134696.13733662 107277.98300152
149798.43194079 209313.31063818 171824.95015212 177306.30228113
179279.44209523 240314.21687073 162365.88465695 164287.20607549
211732.42691019 269298.98627855 240988.74196356 117622.86078148
144873.42966676 130857.00112547 157746.22385322 133994.17539816
196458.0772028 117382.30482051 268833.81608121 149299.72353741
290574.13118432 184139.73018183 193293.04774201 182256.14003094

```
 130866.65558655 264980.73998533 152717.59686868 292969.81664607
 237889.31577873 122654.57415467 148529.32298931 257679.69402859
 152996.59130052 176172.61074251 200104.49499045 205849.64929466
 127898.46904291 124462.83243159  71070.32422949 201412.5855833
 173924.79666344 215665.07176923 189975.46786651 124530.39430917
 227602.80608526 121799.88654278 207053.94611145 109868.58626809
 192019.39694861 163100.4834532  136702.08026562 124790.52673393
 188827.33398998 175625.49377332 138423.16219231 234585.07250701
 162884.62283656 114239.91169223 242366.88154827 139457.12915295
 163334.66791998 134997.81677407 171540.15542348 137137.20478442
 168964.95572282 140891.2750795  146253.03620753 271427.67657468
 184447.83442726 135698.00012238 116439.07035082 147315.93664796
 139394.79448387 126999.1354553  164399.03315589 134882.36954128
 197145.86168894 150049.85192589 126781.39270585 165003.40101427
 108430.29685228 301800.57077443  98773.33738733 120384.92970781
  98751.80097918 130793.49020116 158368.06404164 141610.1061013
 277480.19794627 159914.30263346 173992.20059838]
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
<class 'numpy.float64'>
Linear Regression 0.8866111199235576
Logistic Regression 0.0029154518950437317
```

```
       ␣
 ↪---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call␣
 ↪last)

        <ipython-input-35-16398a4e30c4> in <module>
         20
         21 from sklearn.metrics import classification_report
    ---> 22 classification_report(y_test.values, y_pred)


        ~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in␣
 ↪inner_f(*args, **kwargs)
         71                           FutureWarning)
         72         kwargs.update({k: arg for k, arg in zip(sig.parameters,␣
 ↪args)})
    ---> 73         return f(**kwargs)
         74     return inner_f
         75
```

```
      ~\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py in␣
↪classification_report(y_true, y_pred, labels, target_names, sample_weight,␣
↪digits, output_dict, zero_division)
     1927     """
     1928
   -> 1929     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
     1930
     1931     labels_given = True


      ~\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py in␣
↪_check_targets(y_true, y_pred)
       88
       89     if len(y_type) > 1:
   ---> 90         raise ValueError("Classification metrics can't handle a mix␣
↪of {0} "
       91                          "and {1} targets".format(type_true,␣
↪type_pred))
       92


      ValueError: Classification metrics can't handle a mix of multiclass and␣
↪continuous targets
```