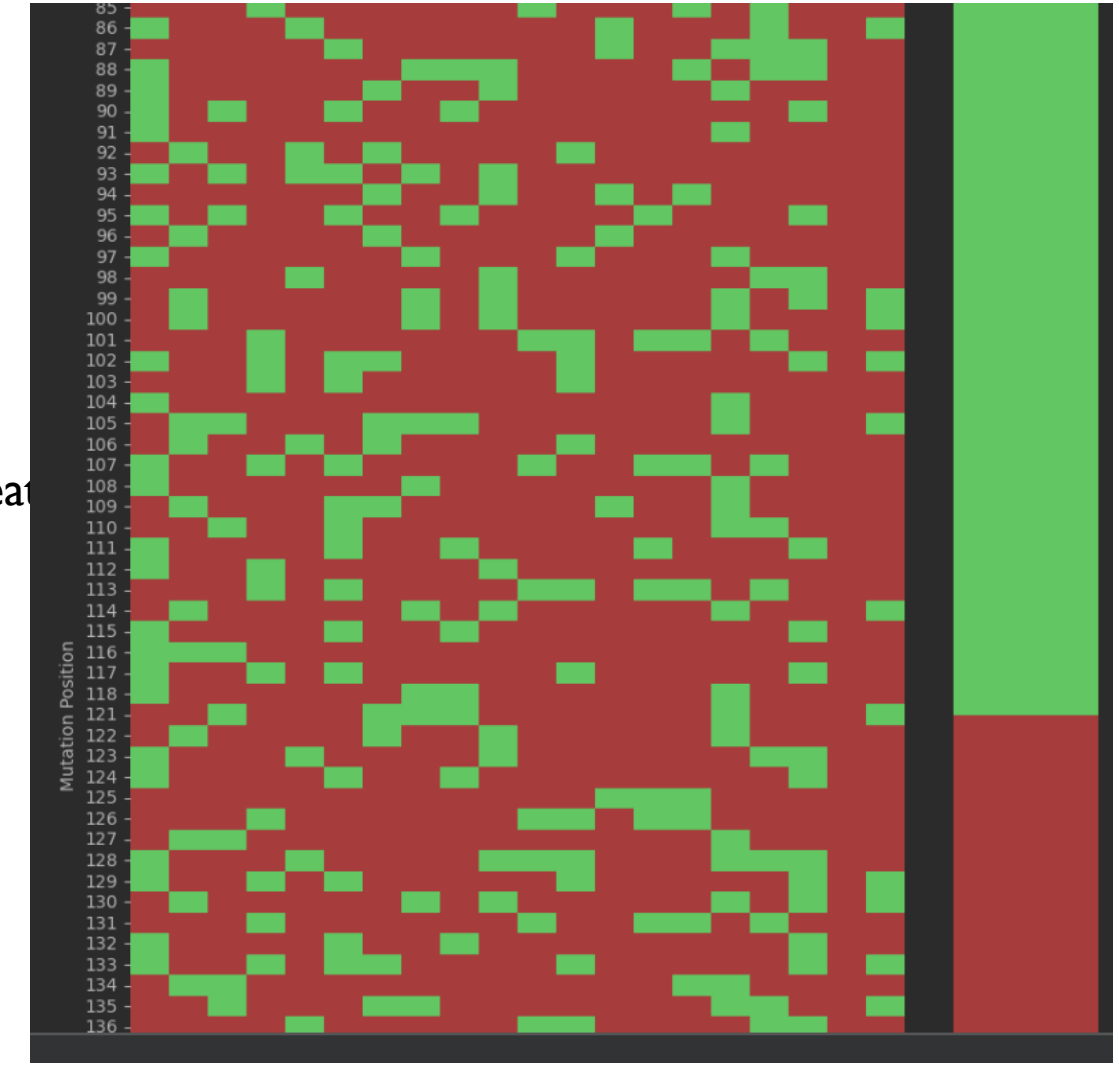
DMS TUT 07.06.23

Gruppe: GFP

ROMAN 0

- Basic analysis
  - Mean und Median der Fitness-scores basierend auf individuellen AS und Positionen
  -
  -
- Heat

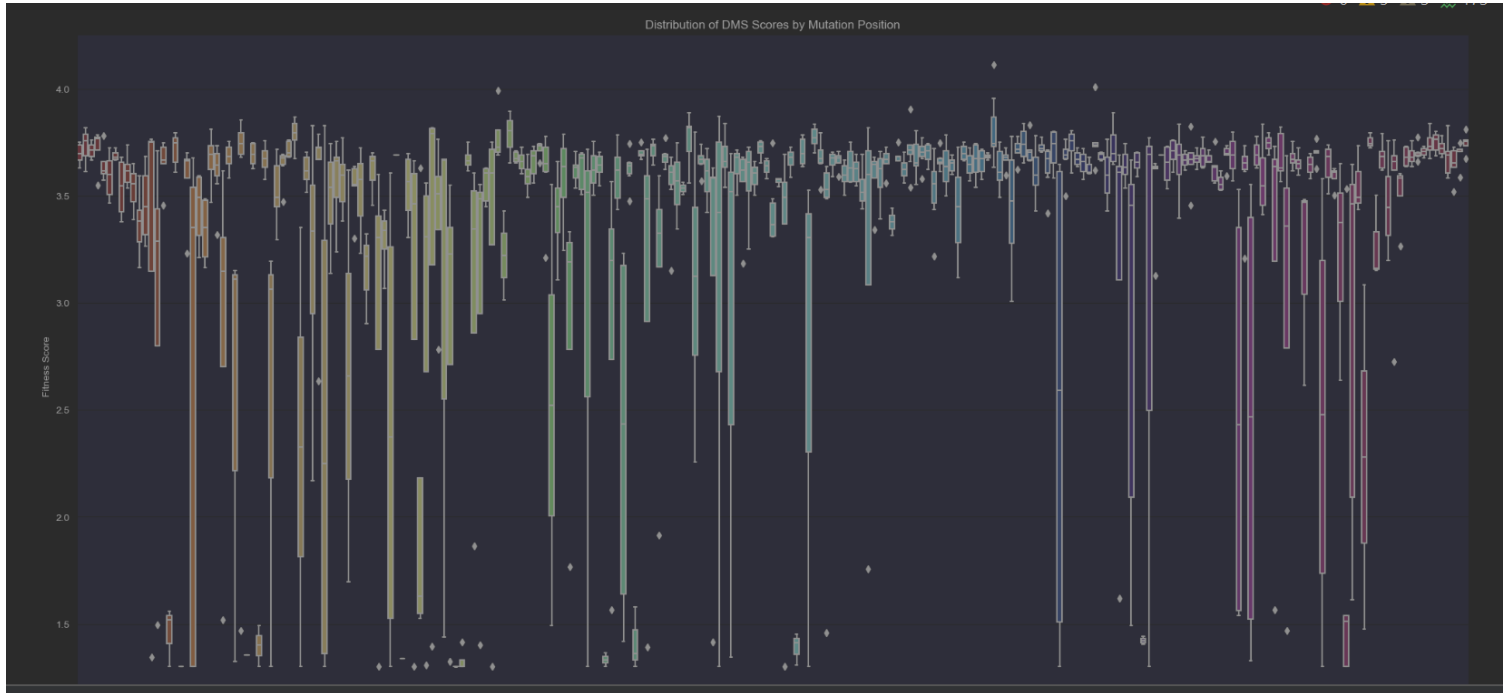


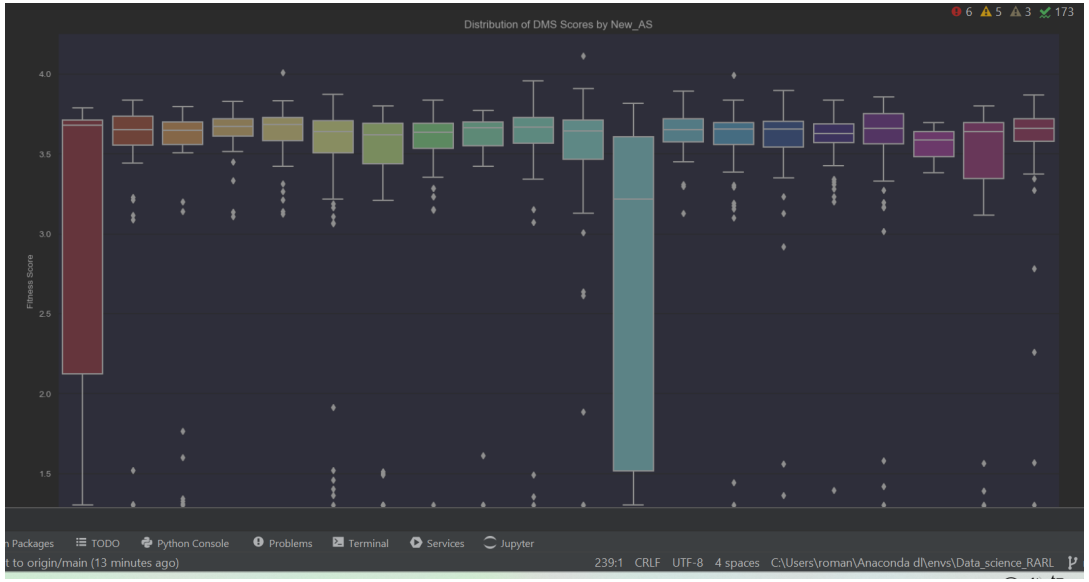
Anova = t-test für viele Gruppen

- ANOVA für Position vs. Fitness-score
- ANOVA für AS vs. Fitness-score
- --> Aber nicht normalverteilt.....A

ROMAN 0.5

Scores gegen Position





Scores gegen AS

- Eta-Quadrat sagt aus, welche Effektstärke.
- Basiert auf ANOVA --> ungültig?
- Quotient der Quadratsumme des Modells und der totalen Quadratsumme

ROMAN 0.75

Bonferroni correction is a conservative correction method that controls the family-wise error rate (FWER) by dividing the desired significance level (e.g., 0.05) by the number of comparisons.

ANOVA aus allen 3:

	df	sum_sq	mean_sq	F	PR(>F)
Position	232.0	243.851576	1.051084	6.003194	1.301208e-82
New_AS	19.0	29.717435	1.564076	8.933106	1.507908e-23
Residual	832.0	145.672832	0.175088	NaN	NaN

Eta-Squared Position
0.5816489450614979
58% der Varianz

Eta-Squared AS
0.07088375323670729
7% der Varianz

AS-Wert kommt hin mit Boxplot?

Bonferroni-Correction --> Keine gravierende Änderung

	Corrected P-value
Position	3.903625e-82
New_AS	4.523723e-23
Residual	NaN

ROMAN 1

Es geht an die statistischen Tests!

SHAPIRO-WILK

```python
#Test, ob Normalverteilt
from scipy.stats import shapiro
# Shapiro-Wilk-Test
stat, p = shapiro(Roman_1["Fitness_Score"])
alpha = 0.05

if p > alpha:
    print("Daten sind normalverteilt (H0 nicht ablehnen)")
else:
    print("Daten sind nicht normalverteilt (H0 ablehnen)")
print()
print(p)
```

Executed at 2023.06.06 18:59:07 in 34ms

```
Daten sind nicht normalverteilt (H0 ablehnen)

0.0
```

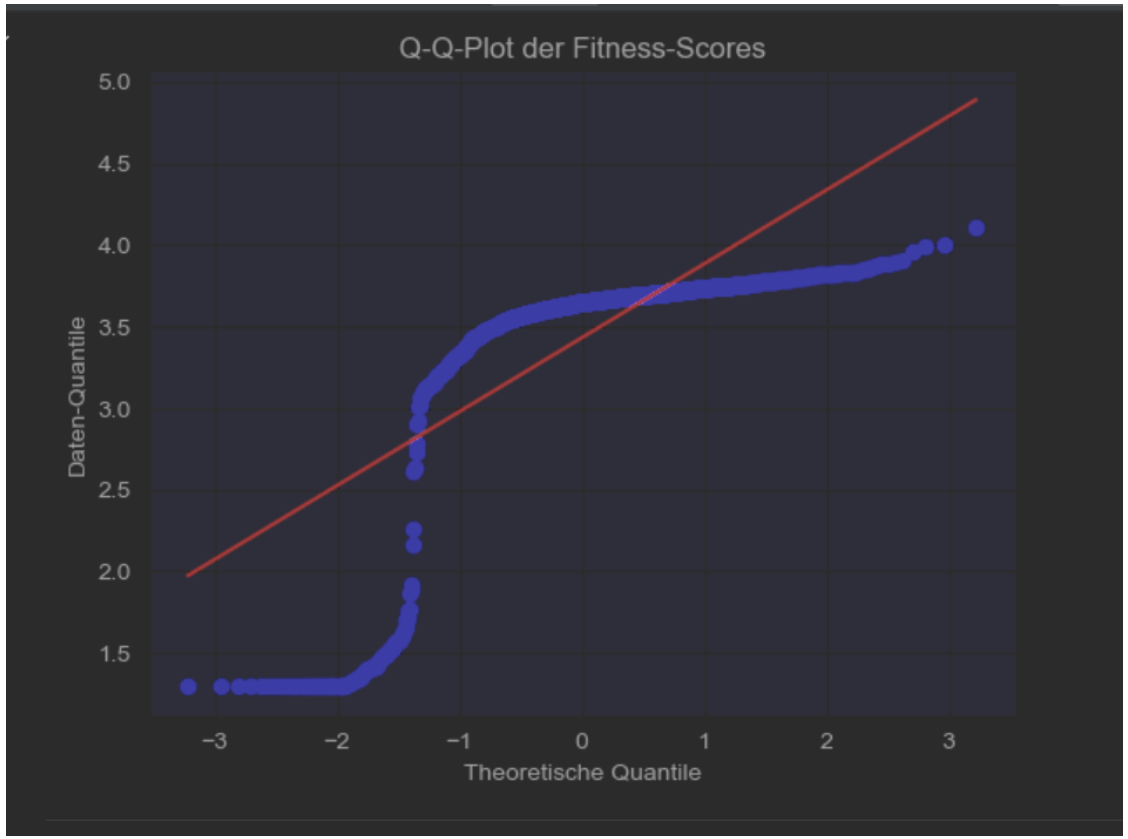
SHAPIRO-WILK

- Testet, ob normalverteilt
- Quotient, der das Verhältnis zweier Varianz-Schätzer zueinander ausdrückt
- Schätzer im Zähler: erwartete Varianz (wenn aus einer Normalverteilung)
- Schätzer im Nenner: tatsächliche Varianz der Stichprobe

ROMAN 2

Es geht an die statistischen Tests!

Q-Q-Plot



ROMAN 3.1

Es geht an die statistischen Tests!

MANN-WHITNEY-U

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[5], line 23
     20 fitness_scores = group['Fitness_Score'].values
     22 # Wilcoxon-Rangsummentest
---> 23 statistic, p_value = mannwhitneyu(fitness_scores, alternative='two-sided')
     25 alpha = 0.05  # Signifikanzniveau
     27 print(f"Position: {position}, Aminosäure: {amino_acid}")
```

- KeyError 'y'
- Auch bei der Verwendung von ranksums als Funktion
- Ein Argument hat gefehlt?
- Gelöst durch zusätzliches Argument (irgendwie?)

```
statistic, p_value = stats.mannwhitneyu(fitness_scores_position, fitness_scores_AS, alternative='two-sided')
```

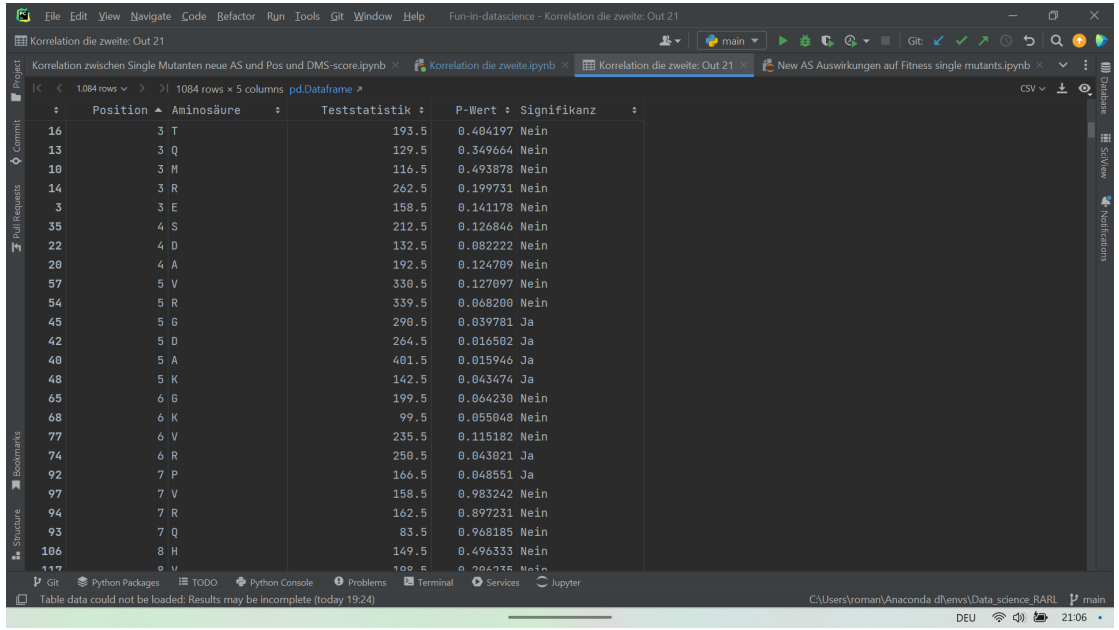
- Mit Maske alle rausgepickt, die tatsächlich existieren
  - Er hat alle * alle gemacht
  - Muss ich ihm vorher sagen, dass er die entfernen soll wegen Datenverzerrung?

- Wenn verglichen mit Diagramm von früher --> kommt oft umgekehrt hin
  - Ja = guter DMS-score
  - Nein = schlechter score
- Wenn es nicht hinkommt, dann ….?

ROMAN 3.2

Es geht an die statistischen Tests!

MANN-WHITNEY-U

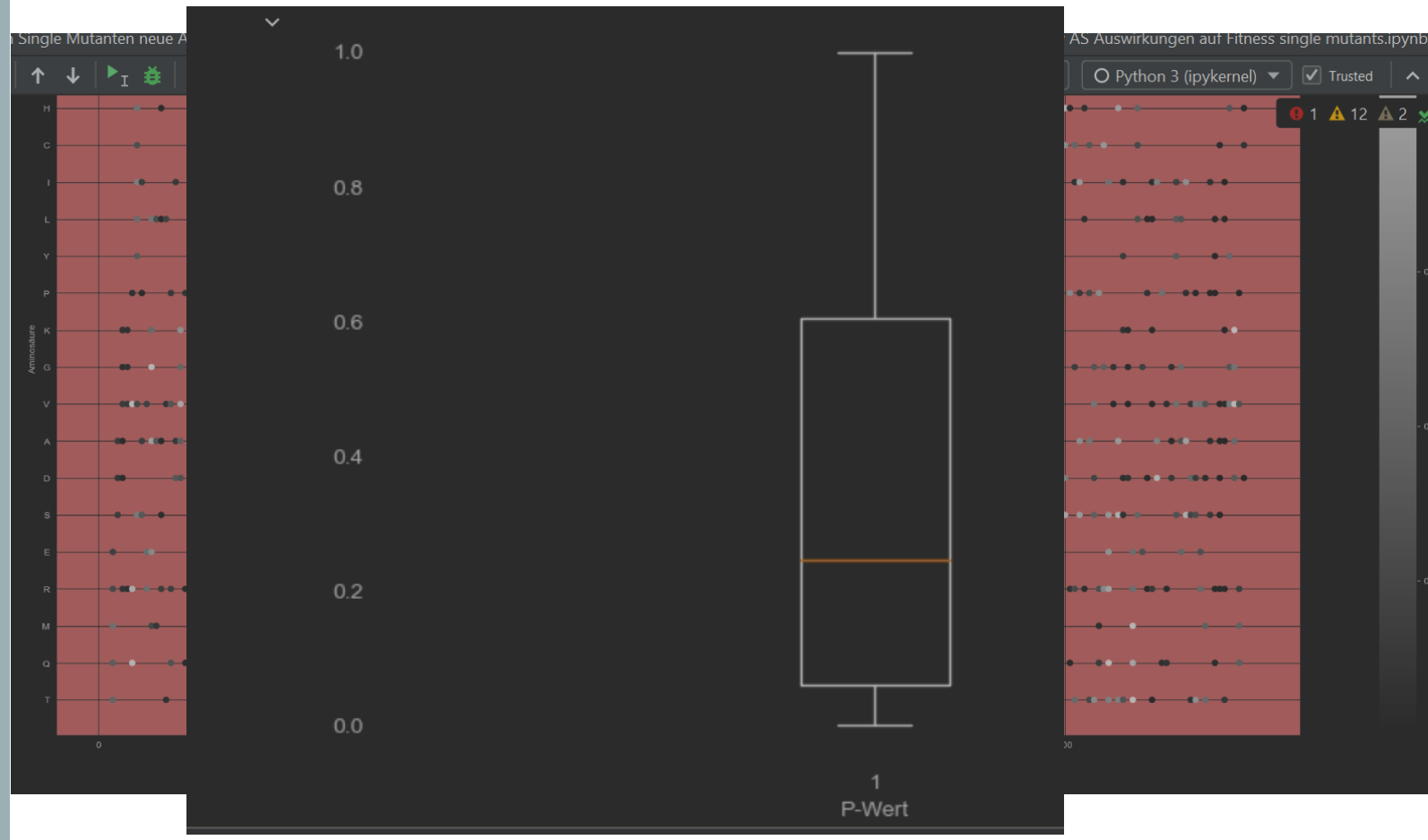


	Position	Aminosäure	Teststatistik	P-Wert	Signifikanz
16	3	T	193.5	0.404197	Nein
13	3	Q	129.5	0.349664	Nein
10	3	M	116.5	0.493878	Nein
14	3	R	262.5	0.199731	Nein
3	3	E	158.5	0.141178	Nein
35	4	S	212.5	0.126846	Nein
22	4	D	132.5	0.082222	Nein
20	4	A	192.5	0.124709	Nein
57	5	V	330.5	0.127097	Nein
54	5	R	339.5	0.068200	Nein
45	5	G	290.5	0.039781	Ja
42	5	D	264.5	0.016502	Ja
40	5	A	401.5	0.015946	Ja
48	5	K	142.5	0.043474	Ja
65	6	G	199.5	0.064230	Nein
68	6	K	99.5	0.055048	Nein
77	6	V	235.5	0.115182	Nein
74	6	R	250.5	0.043021	Ja
92	7	P	166.5	0.048551	Ja
97	7	V	158.5	0.983242	Nein
94	7	R	162.5	0.897231	Nein
93	7	Q	83.5	0.968185	Nein
106	8	H	149.5	0.496333	Nein

ROMAN 3.3

Es geht an die statistischen Tests!

MANN-WHITNEY-U



ROMAN 4

Es geht an die statistischen Tests!

KRUSKAL-WALLIS

FRIEDMAN

Kruskal-Wallis-Test
Teststatistik: 577.8469817125756
P-Wert: 1.8399990857612868e-31
Es gibt einen signifikanten Unterschied zwischen den Positionen

Kruskal-Wallis-Test
Teststatistik: 1083.0
P-Wert: 0.494285292349901 04
Es gibt keinen signifikanten Unterschied zwischen den Positionen und Aminosäuren.

Friedman-Test
Teststatistik: 1083.0
P-Wert: 0.494285292349901 04
Es gibt keinen signifikanten Unterschied zwischen den Positionen und Aminosäuren.

Aussagekraft?

ROMAN 5

AS-EIGENSCHAFTEN

1. Kaggle-Dataframe gereinigt
2. Umutierte Sequenz erstellt
3. Currently: Hydrophobicity ganz berechnen
4. Zukunfts-Idee: kleine Nachbarschaft analysieren und Vergleich mit unmutierter Sequenz
   - Für Position
   - Für Aminosäure-Eigenschaft

- Frage: Wie groß sollte man die Nachbarschaft wählen? +/- 4?
- Antwort:
- Frage: Geeigneter Dateityp für diese Analysen? Substrings?
- Antwort:

ANGELA – MÖGLICHE EVOLUTIONSWEGE

Idealfall:

Mutante	m1	m2	m3	m4	m5	m6	m7
I	K3R						
II	K3R	K4R					
III	K3R	K4R	K5R				
IV	K3R	K4R	K5R	K6R			
V	K3R	K4R	K5R	K6R	K7R		

ANGELA – MÖGLICHE EVOLUTIONSWEGE

Realität:

Mutante	m1	m2	m3	m4	m5	m6	m7
I	K5R						
II	K5R						
III	K2R	K4R	K5R				
IV	K1R	K2R	K3R	K4R	K5R		
V	K3R	K4R	K5R				

→ Filtern innerhalb einzelner Spalten → verfälschtes Ergebnis

ANGELA

file: A_ord_Stammbaum

- weiterhin Probleme mit Finden v. sich überschneidenden Mutanten
- mit Listen probiert, aber nichterfolgreich, deshalb wieder mit df gearbeitet

- Verbesserung:
  - Mutationen <u>unabhängig</u> v. Spalte finden u. entsprechende Mutanten rausfiltern✅

als Bsp.: df mit ´K3R´+ weitere beliebige Mutante

```
mutation_1 = 'K3R'
mutation_list = all_possible_mutations
```

```
for mutation in mutation_list:
    # Zeile enthält mind. 1 Element aus mutation_list
    mask_mind_2 = df_mutants_with_specific_mutation.isin
     ([mutation_1, mutation]).sum(axis=1) >= 2
    mask_mutants_with_mutation_1_and_mutation_list |= mask_mind_2
```

ANGELA

- Ansatz: Code erst mit mutation_1 = ‘K3R’ ausprobieren, danach ganze Liste mit allen möglichen Mutationen durchgehen
- → df mit Mutanten, die ‘K3R’ enthalten

- alle weiteren Mutationen zählen
- Welche Mutanten enthalten Mutationen, die im df mehr als x Mal vorkommen (aktuell: x = 12)

```
A110D    1.0
A110S    1.0
A110T    1.0
A110V    1.0
A154S    1.0
        ...
Y92C     1.0
Y92D     1.0
Y92F     1.0
Y92H     2.0
Y92N     1.0
Length: 658,
```

```
I229T    13.0
K113R    15.0
N23S     13.0
```

ANGELA

- neue dfs mit ´K3R´ + mind. 1 häufige (Vorkommen > x) Mutation
- Annahme: 3 dfs der Art f. jede „1. Mutation“
  - → fast 900 dfs⚡
- Wie kann der Code verbessert werden?
- Ist es realistisch, dass ich damit weiterarbeiten kann?

```
I229T
      DMS_score   m1     m2     m3     m4     m5     m6     m7     m8     m9
       \
9      3.499501  K3R  D102G  Y151C  N170D  I229T   None   None   None   None

74     1.428584  K3R   E34V   G35C  S147C  I229T   None   None   None   None

81     3.503790  K3R    E5K  N105D  D190H  I229T   None   None   None   None

119    3.562984  K3R   E17G   K79M  I229T   None   None   None   None   None

120    3.490331  K3R   E17G  V150A  K156R  I229T   None   None   None   None

270    1.301048  K3R   K26R   K41R  S147G  F165S  D216V  I229T   None   None

338    3.724568  K3R    L7R  N144D  I229T   None   None   None   None   None

430    1.301030  K3R   G35D   K79E  K113E  T118I  I152V  F165S  D180G  N212S

436    1.560380  K3R  F100L  F130S  N149D  I229T   None   None   None   None

484    1.631251  K3R   I47N  L137P  V150I  L221Q  I229T   None   None   None

489    3.486355  K3R   I47T  I229T   None   None   None   None   None   None

511    3.360025  K3R   H25R   H77R   K79T  I229T   None   None   None   None

1100   1.301030  K3R    T9A   I14V   F27L   S28C   K79E   H81R  D190E  N212Y
```

ANGELA – IDEEN FÜR ALTERNATIVE CODES + PROBLEME

VON „HINTEN“ ANFANGEN

1. Mutanten mit 15 Mutationen (bzw. etwas weniger, da kaum Mutanten mit 15 Mutationen vorhanden sind) raussuchen
2. Mutanten mit 1/2/3/… Mutationen weniger raussuchen (restliche Mutationen stimmen überein)

- Probleme
  - unrealistisch, dass „vollständige Wege“ mit 15 aufeinander aufbauende Mutationen vorhanden sind
  - schwierig in Loops zu schreiben (?)

ANGELA – IDEEN FÜR ALTERNATIVE CODES + PROBLEME

Mutante	m1	m2	m3	m4	m5
I	K3R				
II	K3R	K4R			
III	K3R	K4R	K5R		
IV	K3R	K4R	K5R	K6R	
V	K3R	K4R	K5R	K6R	K7R

DF MIT POSITION ALS SPALTE UND MUTANTE ALS ZEILE

- Mutationen wären unabhängig v. AnzahlMutation pro Mutante nach Position sortiert
- danach: Zeilen d. dfs nach Mutationen sortieren

- Probleme
  - hintere Positionen werden trotzdem vernachlässigt, da Sortierung bei der 1. Mutation beginnen würde
  - mögliche Lsg.: auch bei Position 2-238 anfangen
  - aber: Sortierung weiterhin von Zahlenreihenfolge abhängig → verfälschtes Ergebnis

LISA

Plan generell:

- Identifizieren von stabilisierenden Mutationen (bzw. zuerst mal Mutationen die einen großen Effekt auf den DMS score haben)

Mutationen: X X X X A

Bleibt immer gleich

Mutationsanzahl bleibt im Vergelich immer gleich

BEISPIEL:

1. Ich suche nach allen Mutanten die K3R beinhalten
2. Alle die 3 Mutationen haben
3. Von denen die Varianz der fscores untersuchen
4. Wenn Varianz klein: wichtige Mutation, wenn groß: uniwchtigere Mutation

mutant
K3R:D102V:H217L
K3R:D117G:V163A
K3R:D129G:V163A:N198Y
K3R:D133A
K3R:D133G:L195Q
K3R:D133G:R168H
K3R:D155V:M233L
K3R:D19A:H77L:K158R
K3R:D19E:E34G:E124K:L178P:F223Y:M233I
K3R:D19E:K26E:S65P:T97S:G134S:L178H:H181L:S2…

Varianz der f_scores dieser Mutanten berechnen → ein Varianzwert → mit anderen vergelichen

LISA

Was habe ich gemacht:

1. Alle vorkommenden Mutationen als Liste herausschreiben
2. Für jede dieser Mutationen checken in welcher der Mutanten sie vorkommt

→ df (Mutanten X alle_möglichen_Mutationen)

3. Theorie mit der Varianz kontrollieren an Beispiel
4. Die Varianz der DMS-scores der Mutanten berechnen die gleich viele Mutationen haben und eine bestimmte Mutation enthalten
5. Schauen mit wievielen Werten die Varianzwerte berechnet wurden → reliability von den Varianzwerten
6. Graph Anzahl_benutzter_Werte X Varianz
7. Den Code verallgemeinern auf alle Mutationen und nacheinander alle Mutationsanzahlen

```
all_possible_mutations = working_dataframe_only_ms.values.flatten().tolist()
all_possible_mutations = list(set(all_possible_mutations))
#hat einen none Wert, weil der auch ein Wert in der Liste ist -> wegmachen
while None in all_possible_mutations:
    all_possible_mutations.remove(None)
print(all_possible_mutations)
```

Executed at 2023.06.06 20:17:48 in 1s 928ms

```
['L137P', 'M78A', 'H217C', 'I47L', 'H25R', 'D102V', 'H199N', 'I188L', 'V22A', '
 'K156N', 'N164V', 'I188N', 'T97P', 'V193G', 'R96L', 'I14L', 'D82H', 'V163G', '
 'H169Y', 'L195K', 'D36E', 'A227N', 'K113G', 'K52G', 'H231L', 'Q80K', 'M233L',
'G116D', 'V224T', 'T118A', 'E95K', 'Q80W', 'N121C', 'K156M', 'L201H', 'S147T',
'S65P', 'H139N', 'T63D', 'F8C', 'H181L', 'Y74H', 'F223I', 'G116C', 'Y92H', 'N23
'E142G', 'I171A', 'D82A', 'K131M', 'K113R', 'S205F', 'E115R', 'Q94P', 'N146V',
'A226P', 'K79M', 'E235K', 'T49P', 'F99L', 'V93D', 'C48F', 'I188F', 'K214R', 'I1
'R168C', 'G228W', 'E5G', 'Y39E', 'E142M', 'C70Y', 'V68A', 'N121R', 'P192S', 'K8
'E111Q', 'I229M', 'V16D', 'N135G', 'I128M', 'S86C', 'H139R', 'L7S', 'T63A', 'T1
'Y92F', 'N159H', 'T97A', 'K158T', 'K156R', 'L60F', 'V193T', 'L201Q', 'Q177E', '
'L178F', 'N164Y', 'G40D', 'G160R', 'E90G', 'K166Q', 'D234N', 'S65L', 'E222Q', '
```

LISA

Was habe ich gemacht:

1. Alle vorkommenden Mutationen als Liste herausschreiben
2. Für jede dieser Mutationen checken in welcher der Mutanten sie vorkommt

→ df (Mutanten X alle_möglichen_Mutationen)

3. Theorie mit der Varianz kontrollieren an Beispiel
4. Die Varianz der DMS-scores der Mutanten berechnen die gleich viele Mutationen haben und eine bestimmte Mutation enthalten
5. Schauen mit wievielen Werten die Varianzwerte berechnet wurden → reliability von den Varianzwerten
6. Graph Anzahl_benutzter_Werte X Varianz
7. Den Code verallgemeinern auf alle Mutationen und nacheinander alle Mutationsanzahlen

CODE:

```python
list_of_dfs = []

# For-Loop zum Erstellen und Hinzufügen eines neuen Columns in jedem Durchlauf
for i in all_possible_mutations:
    new_column_name = f'{i}'
    new_column_values = [only_mutants_list.str.contains(i, regex= False)]
    new_df = pd.DataFrame({new_column_name: new_column_values})
    new_df_exploded = new_df.explode(new_column_name)
    list_of_dfs.append(new_df_exploded)

# Verketten der DataFrames in der Liste
result_how_often = pd.concat(list_of_dfs, axis=1)
result_how_often = result_how_often.reset_index(drop=True)

Executed at 2023.06.06 11:15:09 in 40s 487ms
```

LISA

Was habe ich gemacht:

1. Alle vorkommenden Mutationen als Liste herausschreiben
2. Für jede dieser Mutationen checken in welcher der Mutanten sie vorkommt

→ df (Mutanten X alle_möglichen_Mutationen)

3. Theorie mit der Varianz kontrollieren an Beispiel
4. Die Varianz der DMS-scores der Mutanten berechnen die gleich viele Mutationen haben und eine bestimmte Mutation enthalten
5. Schauen mit wievielen Werten die Varianzwerte berechnet wurden → reliability von den Varianzwerten
6. Graph Anzahl_benutzter_Werte X Varianz
7. Den Code verallgemeinern auf alle Mutationen und nacheinander alle Mutationsanzahlen

RESULT:

C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
L137P	M78A	H217C	I47L	H25R	D102V	H199N	I188L	V22A	L44R
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	True	False	False	False	False
False	False	False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False	False	False

LISA

Was habe ich gemacht:

1. Alle vorkommenden Mutationen als Liste herausschreiben
2. Für jede dieser Mutationen checken in welcher der Mutanten sie vorkommt

→ df (Mutanten X alle_möglichen_Mutationen)

3. Theorie mit der Varianz kontrollieren an Beispiel
4. Die Varianz der DMS-scores der Mutanten berechnen die gleich viele Mutationen haben und eine bestimmte Mutation enthalten
5. Schauen mit wievielen Werten die Varianzwerte berechnet wurden
   → reliability von den Varianzwerten
6. Graph Anzahl_benutzter_Werte X Varianz
7. Den Code verallgemeinern auf alle Mutationen und nacheinander alle Mutationsanzahlen

```
In 14
K166Q_count_fscore = count_fscore_frame.loc[result_how_often['K166Q'] == True]
fscore_2_K166Q = K166Q_count_fscore['DMS_score'].loc[K166Q_count_fscore['mut_count'] == 2]
varianz_K166Q = fscore_2_K166Q.var()
varianz_K166Q
Executed at 2023.06.06 11:15:10 in 358ms

Out 14
0.015426224360537455
```

```
In 15
#vergleich mit einer anderen Mutation
K3R_count_fscore = count_fscore_frame.loc[result_how_often['K3R'] == True]
fscore_2_K3R = K3R_count_fscore['DMS_score'].loc[K3R_count_fscore['mut_count'] == 2]
varianz_K3R = fscore_2_K3R.var()
varianz_K3R
Executed at 2023.06.06 11:15:10 in 347ms

0.2736357484727858
```

K166Q ist in Papern als eine Mutation mit einem der größten Effekte auf den Gesamt-fscore bekannt
→ Sollte (und hat) eine kleinere Varianz in den Werten als eine Mutation am Rand (unwichtiger)

LISA

Was habe ich gemacht:

1. Alle vorkommenden Mutationen als Liste herausschreiben
2. Für jede dieser Mutationen checken in welcher der Mutanten sie vorkommt

→ df (Mutanten X alle_möglichen_Mutationen)

3. Theorie mit der Varianz kontrollieren an Beispiel
4. Die Varianz der DMS-scores der Mutanten berechnen die gleich viele Mutationen haben und eine bestimmte Mutation enthalten
5. Schauen mit wievielen Werten die Varianzwerte berechnet wurden → reliability von den Varianzwerten
6. Graph Anzahl_benutzter_Werte X Varianz
7. Den Code verallgemeinern auf alle Mutationen und nacheinander alle Mutationsanzahlen

```python
variance_per_mutant_2_list = []

for i in all_possible_mutations:
    mut_count_fscore_2 = count_fscore_frame.loc[result_how_often[i] == True]
    fscore_2_mut = mut_count_fscore_2['DMS_score'].loc[mut_count_fscore_2['mut_count'] == 2]
    varianz_mut_2 = fscore_2_mut.var()
    variance_per_mutant_2_list.append(varianz_mut_2)

variance_per_mutant_2_series = pd.Series(variance_per_mutant_2_list, index=all_possible_mutations)
variance_per_mutant_2_df = variance_per_mutant_2_series.to_frame()

print(variance_per_mutant_2_df)
```

Executed at 2023.06.06 11:15:16 in 6s 574ms

```
               0
L137P  0.029233
M78A        NaN
H217C       NaN
I47L   0.020612
H25R   0.096180
...         ...
D155V  0.016701
L44M   0.001068
K166G       NaN
T43N        NaN
Y200D  0.003056
```

LISA

Was habe ich gemacht:

1. Alle vorkommenden Mutationen als Liste herausschreiben
2. Für jede dieser Mutationen checken in welcher der Mutanten sie vorkommt

→ df (Mutanten X alle_möglichen_Mutationen)

3. Theorie mit der Varianz kontrollieren an Beispiel
4. Die Varianz der DMS-scores der Mutanten berechnen die gleich viele Mutationen haben und eine bestimmte Mutation enthalten
5. Schauen mit wievielen Werten die Varianzwerte berechnet wurden → reliability von den Varianzwerten
6. Graph Anzahl_benutzter_Werte X Varianz
7. Den Code verallgemeinern auf alle Mutationen und nacheinander alle Mutationsanzahlen

```python
#rausfinden wie viele Daten wir jeweils haben zum berechnen
how_many_for_variance_2 = []

for i in all_possible_mutations:
    mut_count_fscore_2 = count_fscore_frame.loc[result_how_often[i] == True]
    fscore_2_mut = mut_count_fscore_2['DMS_score'].loc[mut_count_fscore_2['mut_count'] == 2]
    wie_viel_jeweils_2 = len(fscore_2_mut)
    how_many_for_variance_2.append(wie_viel_jeweils_2)

how_many_for_variance_2 = pd.Series(how_many_for_variance_2, index=all_possible_mutations)
how_many_for_variance_2_df = how_many_for_variance_2.to_frame()

print(how_many_for_variance_2_df)
```

Executed at 2023.06.06 11:15:22 in 5s 985ms

```
         0
L137P   16
M78A     0
H217C    0
I47L     6
H25R    66
...     ..
D155V   19
L44M     4
```

LISA

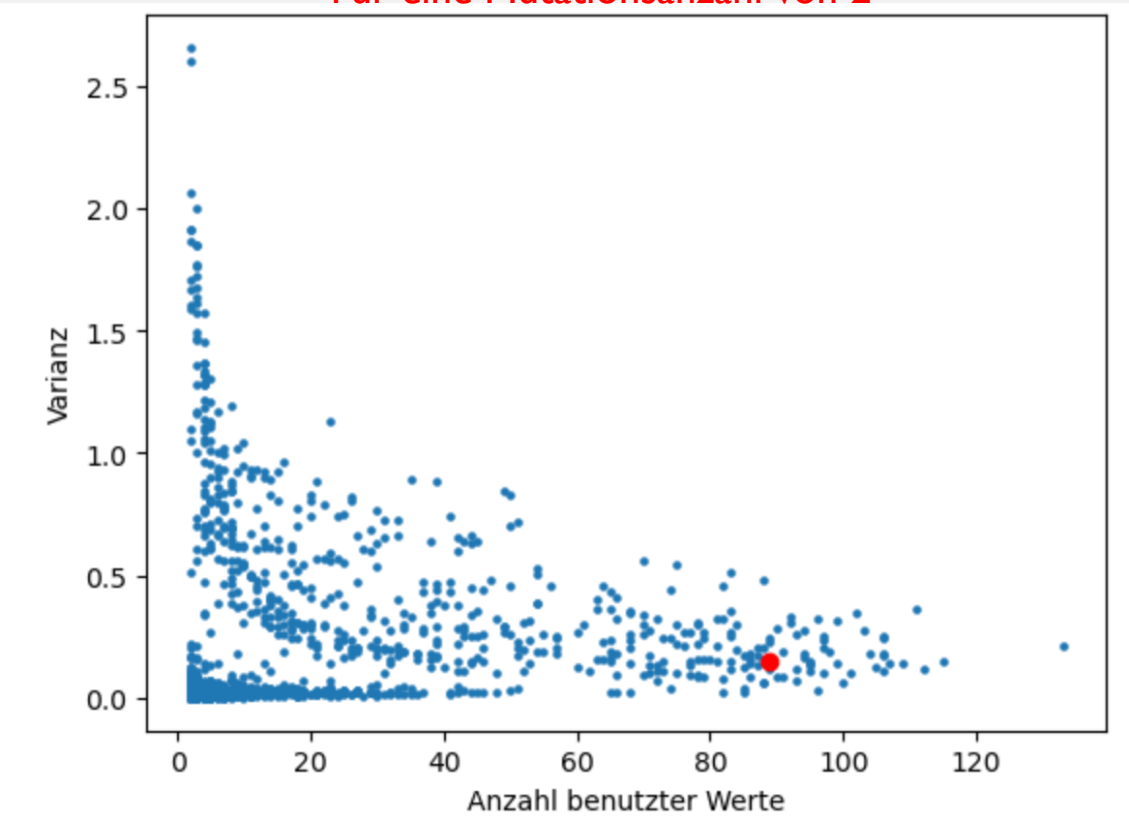
Was habe ich gemacht:

1. Alle vorkommenden Mutationen als Liste herausschreiben
2. Für jede dieser Mutationen checken in welcher der Mutanten sie vorkommt

→ df (Mutanten X alle_möglichen_Mutationen)

3. Theorie mit der Varianz kontrollieren an Beispiel
4. Die Varianz der DMS-scores der Mutanten berechnen die gleich viele Mutationen haben und eine bestimmte Mutation enthalten
5. Schauen mit wievielen Werten die Varianzwerte berechnet wurden → reliability von den Varianzwerten
6. Graph Anzahl_benutzter_Werte X Varianz
7. Den Code verallgemeinern auf alle Mutationen und nacheinander alle Mutationsanzahlen

Für eine Mutationsanzahl von 2



LISA

Was habe ich gemacht:

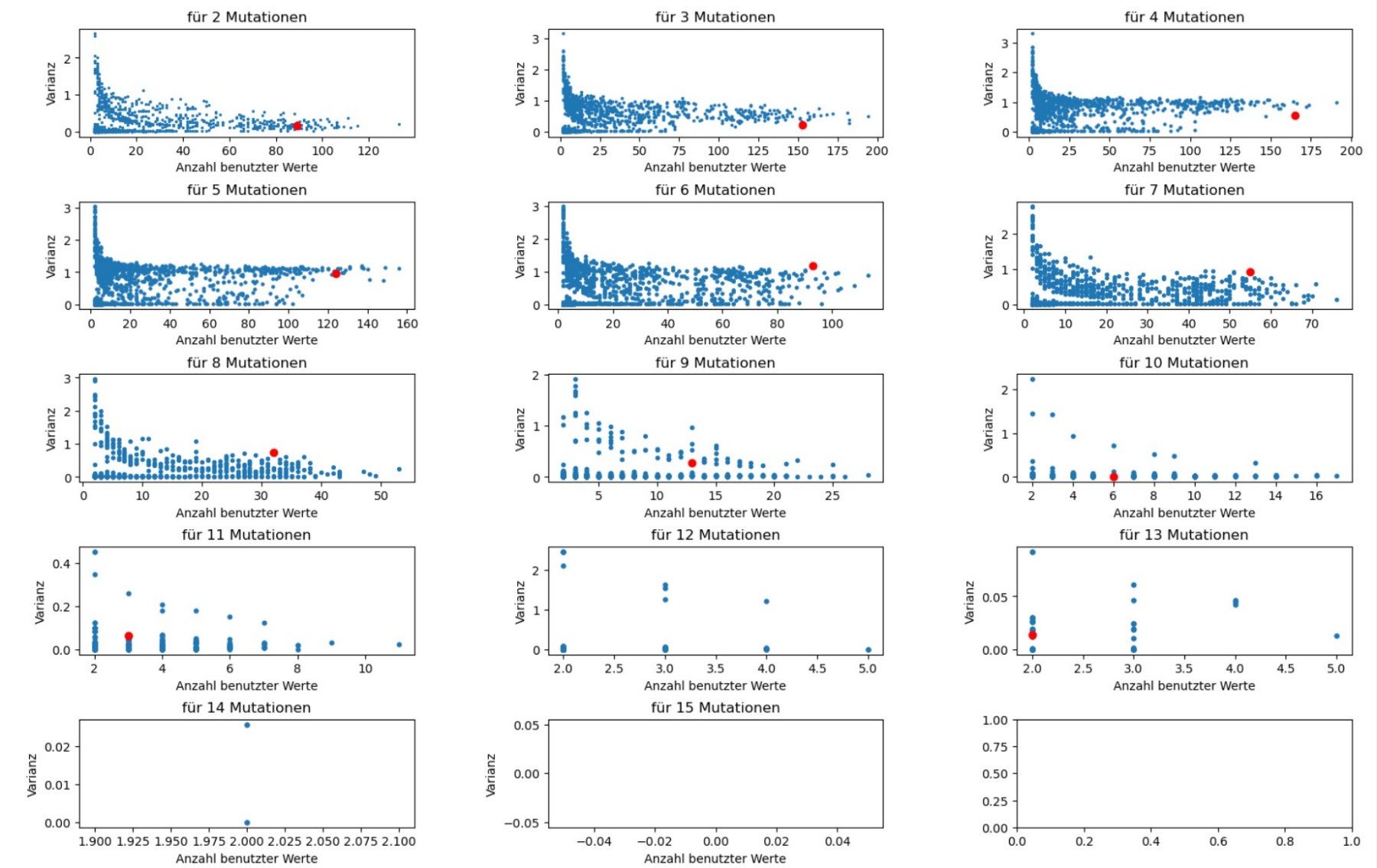
1. Alle vorkommenden Mutationen als Liste herausschreiben
2. Für jede dieser Mutationen checken in welcher der Mutanten sie vorkommt

→ df (Mutanten X alle_möglichen_Mutationen)

3. Theorie mit der Varianz kontrollieren an Beispiel
4. Die Varianz der DMS-scores der Mutanten berechnen die gleich viele Mutationen haben und eine bestimmte Mutation enthalten
5. Schauen mit wievielen Werten die Varianzwerte berechnet wurden → reliability von den Varianzwerten
6. Graph Anzahl_benutzter_Werte X Varianz
7. Den Code verallgemeinern auf alle Mutationen und nacheinander alle Mutationsanzahlen

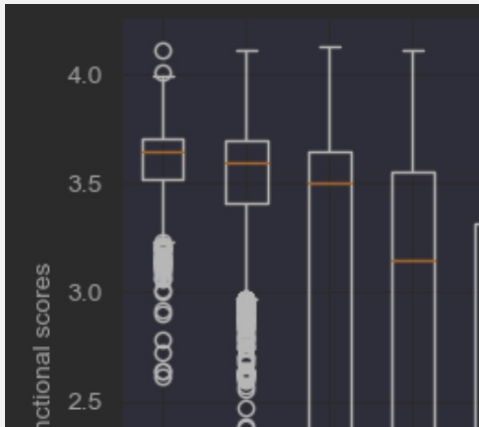
```python
#verallgemeinert
import matplotlib.pyplot as plt


fig, axes = plt.subplots(nrows=5, ncols=3, figsize=(19, 12))  # Abbildung und Achsenobjekte erstellen
plt.subplots_adjust(wspace=0.4, hspace=0.6)

for j, ax in zip(range(2, 16), axes.flatten()):
    variance_per_mutant_list = []

    for i in all_possible_mutations:
        mut_count_fscore = count_fscore_frame.loc[result_how_often[i] == True]
        fscore_mut = mut_count_fscore['DMS_score'].loc[mut_count_fscore['mut_count'] == j]
        varianz_mut = fscore_mut.var()
        variance_per_mutant_list.append(varianz_mut)

    variance_per_mutant_series = pd.Series(variance_per_mutant_list, index=all_possible_mutations)
    variance_per_mutant_df = variance_per_mutant_series.to_frame()


#rausfinden wie viele Daten wir jeweils haben zum berechnen
    how_many_for_variance = []

    for i in all_possible_mutations:
        mut_count_fscore = count_fscore_frame.loc[result_how_often[i] == True]
        fscore_mut = mut_count_fscore['DMS_score'].loc[mut_count_fscore['mut_count'] == j]
         wie_viel_jeweils = len(fscore_mut)
         how_many_for_variance.append(wie_viel_jeweils)

     how_many_for_variance = pd.Series(how_many_for_variance, index=all_possible_mutations)
     how_many_for_variance_df = how_many_for_variance.to_frame()


     how_many_AND_variance_df = pd.concat([how_many_for_variance_df, variance_per_mutant_df], axis = 1)
     how_many_AND_variance_df.columns = ['Anzahl benutzter Werte', 'Varianz']
     how_many_AND_variance_df = how_many_AND_variance_df.dropna()

 #scatter plot erstellen, mit benennungen
     ax.scatter(how_many_AND_variance_df['Anzahl benutzter Werte'],how_many_AND_variance_df['Varianz'], s
       = j )
     ax.set_xlabel('Anzahl benutzter Werte')
     ax.set_ylabel('Varianz')

     if "V163A" in how_many_AND_variance_df.index:
         ax.scatter(how_many_AND_variance_df['Anzahl benutzter Werte']['V163A'],
          how_many_AND_variance_df['Varianz']['V163A'], c='red')
     ax.set_title(f'für {j} Mutationen')
 # Diagramm anzeigen

 plt.show()
```

RESULT:



Nach vorigen Untersuchungen (boxplot + tests von letztem Mal) → kann man ab 7/8 einfach ausschließen

Jeder Punkt ist eine Mutation



FRAGE: ab welcher Anzahl von benutzten Werten macht es Sinn die Varianz ernstzunehmen?
→ Nach welchem Kriterium?

LISA – PLAN BIS NÄCHSTE WOCHE

1. **Versuchen ein Ranking von Mutationen zu erstellen**

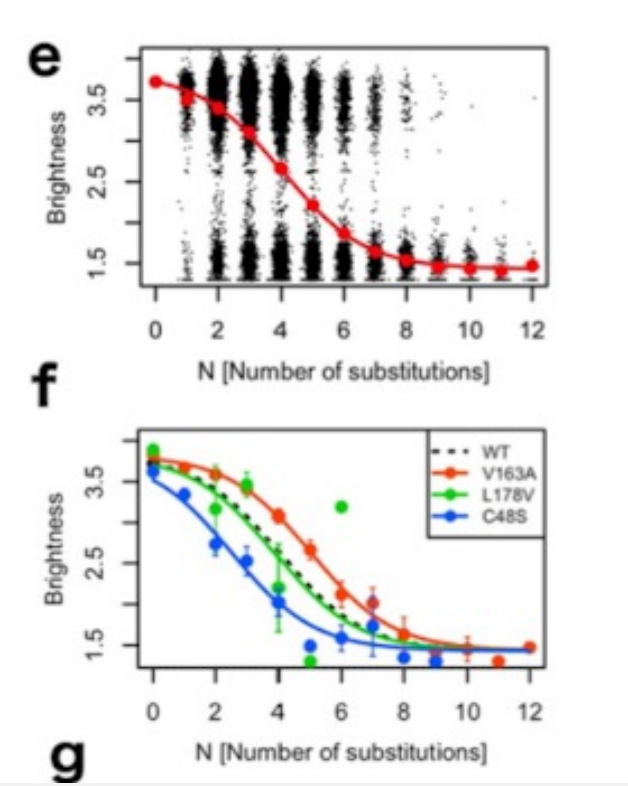
→ Je mehr Daten und desto geringer die Varianz desto besser (je weiter rechts unten im Graph desto besser)

→ Erste Orientierung welche Mutationen wichtig sein könnten

2. **Graph pro Mutation: fitnessscores gegen die Anzahl der Mutationen**

→ Mutationen vergleichen

→ Mit anderem ranking abgleichen (besondere Mutationen aus ranking 1 hier untersuchen)



Aus paper zur Untersuchung von dms Daten an fluoreszenzproteinen