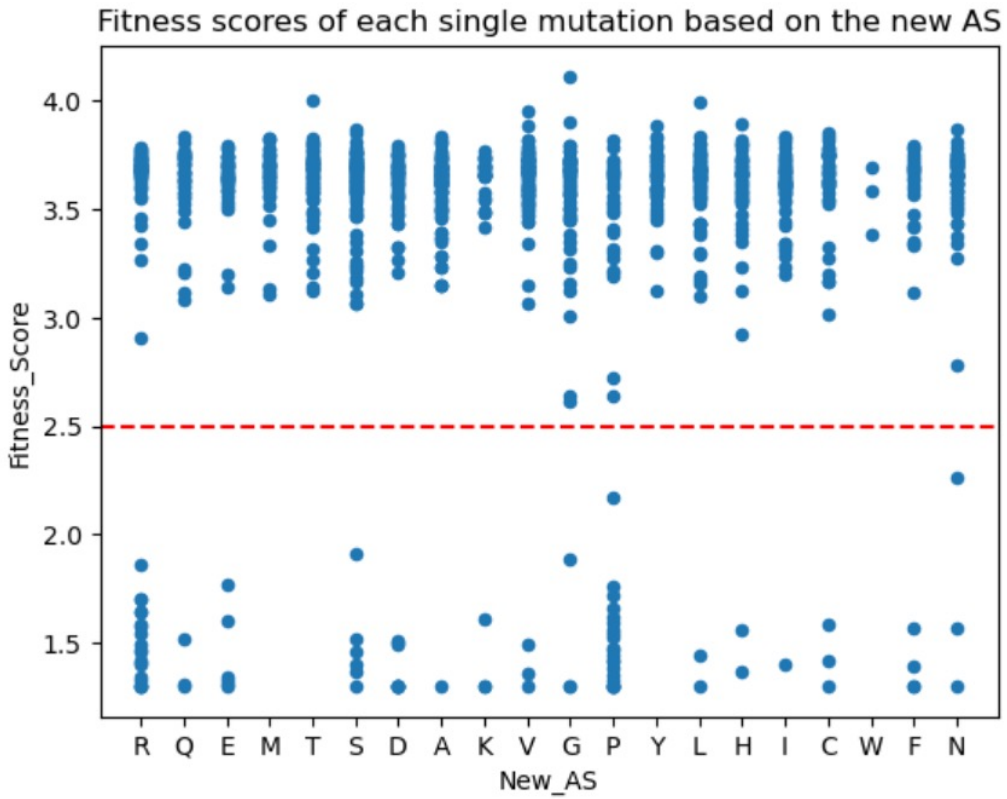
DMS TUT 01.06

Gruppe: GFP

ROMAN 1

Single-Mutanten:

Was ich neues gemacht habe



In [101…

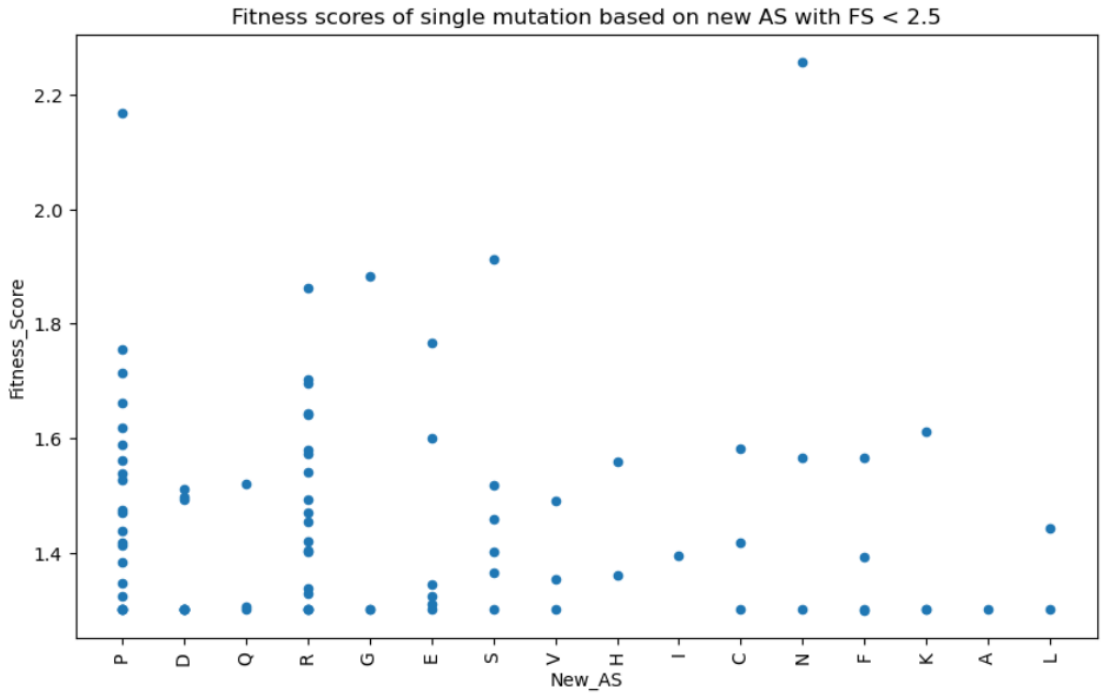
```
import matplotlib.pyplot as plt
dms_score_filtered_newAS.plot(x="New_AS", y="Fitness_Score", kind="scatter")
plt.title("Fitness scores of each single mutation based on the new AS")
threshold = 2.5
plt.axhline(threshold, color='red', linestyle='--', label='Threshold')
plt.show()
#Plot zeigt alle Singlemutanten (neue AS) mit dem jeweiligen Fitness score. Man sieht eine eindeutige Trennung bei 2.5
```

+Auf einzelne neue AS und spezifische Positionen (65-67) geschaut

ROMAN 2

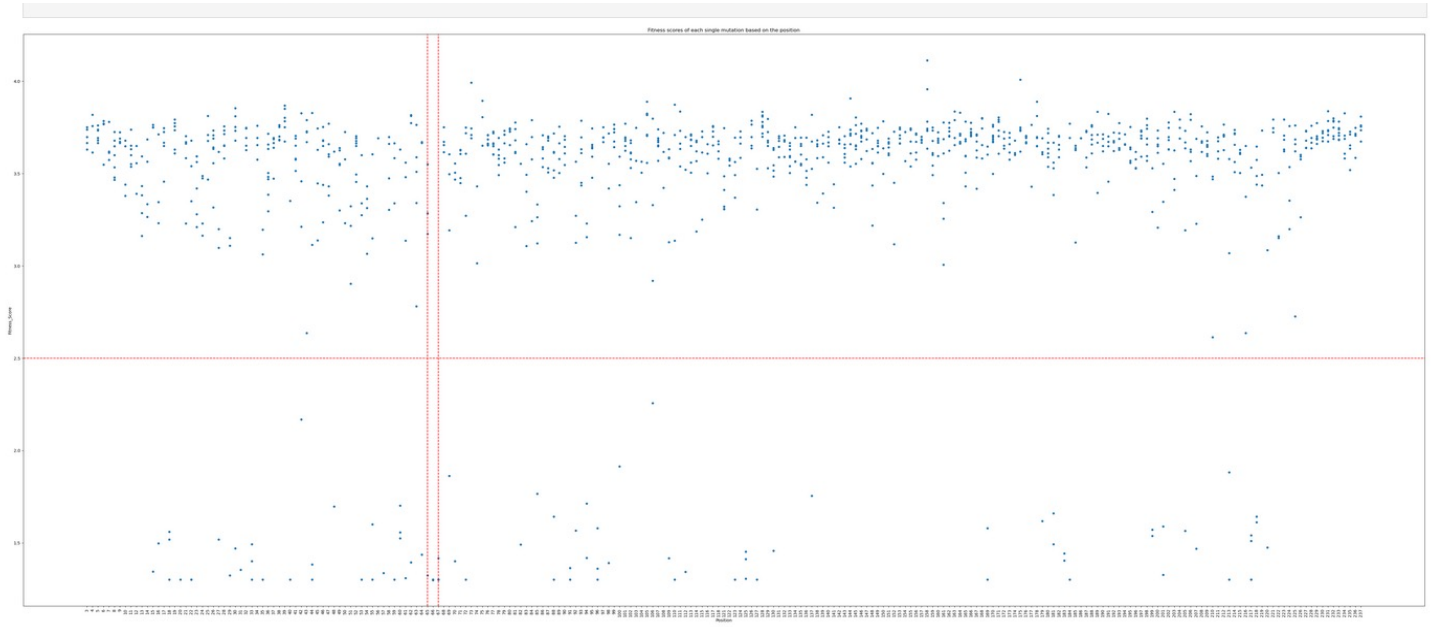
Frage: Wie kann ich die Korrelation von neuen AS zu Position ziehen?

Antwort: Erstmal mit Nachbarschaften arbeiten



ROMAN 3

Frage: Wie kann ich die Achsen besser lesbar machen?
Antwort: Achsenbeschriftung reduzieren (z.B. nur alle 10 Pos)



[1084 rows x 2 columns]

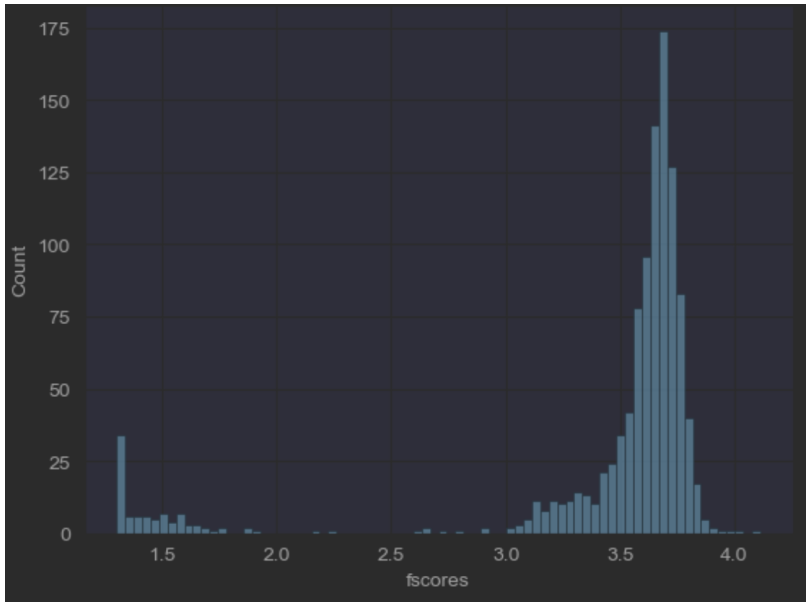
```
import matplotlib.pyplot as plt
mutations_pos_df_mit_scores.plot(x="Position", y="Fitness_Score", kind="scatter")
plt.title("Fitness scores of each single mutation based on the position")
#plt.figure(figsize=(100, 6))
#Macht das Diagramm auf Größe "Breite, Höhe" größer

#plt.xticks(rotation=45)  #
##Rotate the x-axis labels by 45 degrees

plt.xticks(rotation='vertical')
# Rotate the x-axis labels vertically
plt.gcf().set_size_inches(60, 25)
#Increase the width to 10 inches and height to 6 inches

a_threshold = 2.5
plt.axhline(threshold, color='red', linestyle='--', label='Threshold')
a_threshold_65 = 62
plt.axvline(a_threshold_65, color='red', linestyle='--', label='Threshold_65')
a_threshold_67 = 64
plt.axvline(a_threshold_67, color='red', linestyle='--', label='Threshold_67')
#Positionen 65-67 gehören zum Chromophor
#Interessant ist hier, dass es Mutationen gibt, die trotz Mutation im Chromophor an Position 65 einen guten score haben. Pos
plt.show()

#Plot mit single Mutanten basierend auf Position der Mutation.
```

ROMAN 4

- Next step: Wie kann ich AS-Eigenschaften numerische Werte zuordnen?
  - Polar/unpolar --> 1/0
  - Pos geladen/neg geladen --> 1/-1
  - Größe --> ?
  - Wechselwirkungsmöglichkeiten --> ?
  - BENE hat eine Liste geschickt
- Ziel: "Mutation ist an dieser Position schlecht, weil..." beantworten

Datei: statistical tests fscores.ipynb

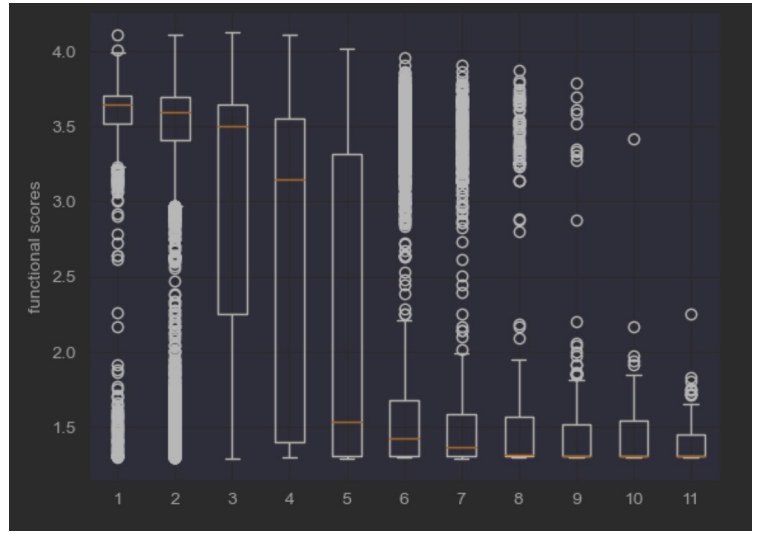
LISA 1: T-TEST VERBESSERUNG VON LETZTEM MAL



Wie die verteilungen mehr oder weniger alle aussehen, aber doch recht verschieden

```
In 16   1  print(np.var(one_mutation_fscores))
        2  print(np.var(two_mutation_fscores))
        3  print(np.var(three_mutation_fscores))
           Executed at 2023.05.29 19:38:37 in 159ms

           fscores          0.386754
           mutation_count   0.000000
           dtype: float64
           fscores          0.480284
           mutation_count   0.000000
           dtype: float64
           fscores          0.862342
           mutation_count   0.000000
           dtype: float64
```

Varianzen von den ersten drei Gruppen untersucht, sind verschieden -> independent t-test war die falsche Wahl



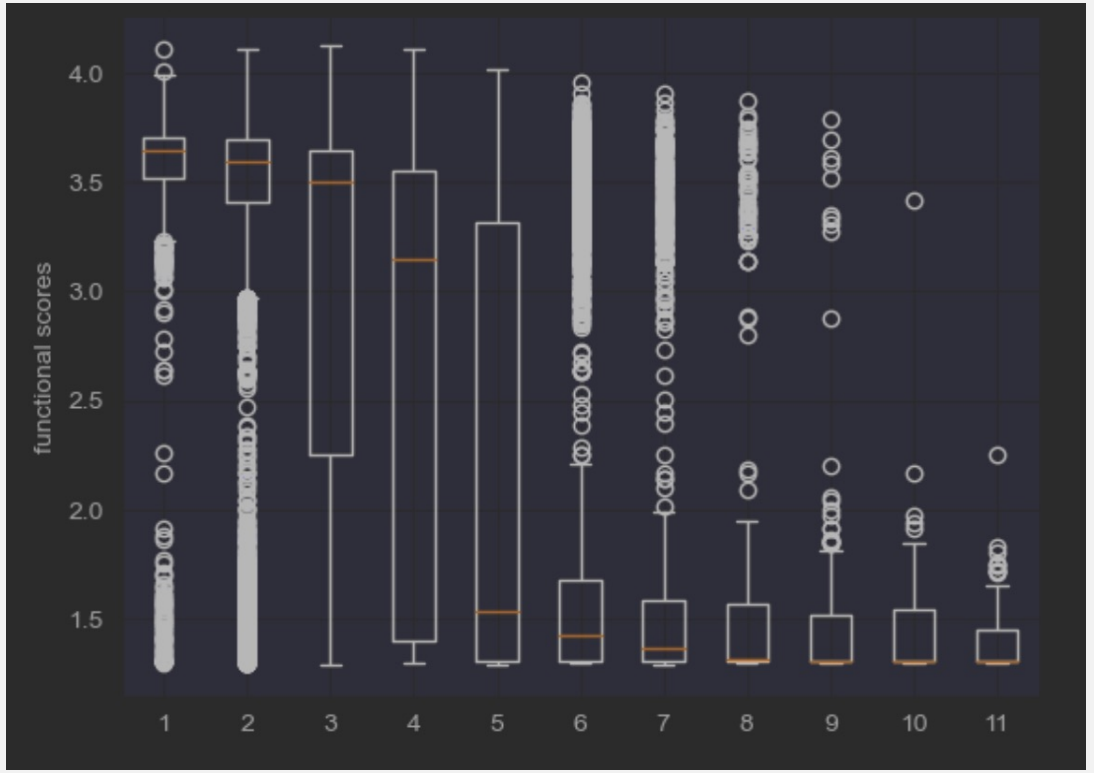
Boxplots für die einzelnen Verteilungen
-> Visualisierung ab wo es „halbwegs ähnlich“ wird

Datei: statistical tests fscores.ipynb

LISA 2: T-TEST VERBESSERUNG VON LETZTEM MAL

- Letztes Mal: Vergleichen der Verteilungen von fitness scores über die Mutationsanzahl
- t-test war nicht die richtige Herangehensweise: weil Varianzen nicht gleich + nicht normalverteilt
- Mann- Whitney- U –test gemacht
- Bei p = 0.05 gelten alle Verteilungen als signifikant verschieden
- Bei p = 0.01 gelten nur noch die Verteilungen von 1 bis 7 als verschieden, ab dem Vergleich von 7 zu 8 gelten sie als nicht mehr signifikant
- -> passt mit dem boxplot von vorher

```
In 73
# mann- whitney-u-test weil nicht normalverteilt, unabhängig
# um zu schauen ob sich die Verteilungen unterscheiden
from scipy.stats import mannwhitneyu
statistic, p_value = mannwhitneyu(eight_mutation_fscores, seven_mutation_fscores)
if p_value[0] > 0.01:
    print('kein signifikanter Unterschied')
else:
    print('signifikanter Unterschied')

Executed at 2023.05.29 20:29:14 in 38ms

kein signifikanter Unterschied
```



Datei: L_second_try_stabilizing_mutations.ipynb

LISA 3: ANFANG STABILISIERENDE MUTATIONEN

- Plan:

1. Angelas Code in meinen übersetzen
2. aus Angelas Code alle möglichen Mutationen rausschreiben, aber nur einmal
3. --> Liste mit allen vorkommenden Mutationen
4. Funktion schreiben, die für jede dieser (einzeln!) alle Sequenzen raussucht in der diese vorkommt
5. --> ein dataframe, mit mutation als spaltenname und allen sequenzen wo es drin vorkommt als Reihen drunter und noch einer Spalte mit mutcount dahinter, pro Mutation werden zwei Spalten an den dataframe angehängt
6. (dadurch können Reihen spezifisch nach mutcount angeprochen werden)
7. innerhalb einer Mutationsspalte, alle mit dem selben mutcount: fscore vergleich : Varianz der Werte als Maß für impact

- -> je unwichtiger die Mutation, desto weniger macht sie was beim fscore, desto weniger sollten sich die Werte unterscheiden, desto kleiner sollte die Varianz sein

Out[17]:

	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12	m13	m14	m15
0	K3R	V55A	Q94R	A110T	D117G	M153K	D216A	None	None	None	None	None	None	None	None
1	K3Q	V16A	I167T	L195Q	None	None	None	None	None	None	None	None	None	None	None
2	K3Q	Y143C	N164D	S205P	A227T	None	None	None	None	None	None	None	None	None	None
3	K3Q	Y143N	V193A	None	None	None	None	None	None	None	None	None	None	None	None
4	K3R	None	None	None	None	None	None	None	None	None	None	None	None	None	None

1. alle Werte des DataFrames mit values.flatten() in eine eindimensionale Liste umgewandeln
2. tolist() verwendet, um die Liste in eine normale Python-Liste umzuwandeln
3. Liste in ein Set umgewandeln (nur eindeutige Elemente)
4. set zurück in Liste

In [6]:

```
all_possible_mutations = working_dataframe_only_ms.values.flatten().tolist()
all_possible_mutations = list(set(all_possible_mutations))
```

Datei: L_second_try_stabilizing_mutations.ipynb

LISA 4: ANFANG STABILISIERENDE MUTATIONEN

- Plan:

1. Angelas Code in meinen übersetzen
2. aus Angelas Code alle möglichen Mutationen rausschreiben, aber nur einmal
3. --> Liste mit allen vorkommenden Mutationen
4. Funktion schreiben, die für jede dieser (einzeln!) alle Sequenzen raussucht in der diese vorkommt
5. --> ein dataframe, mit mutation als spaltenname und boolians als Angabe ob sie in der jeweiligen Mutante(Zeile) vorkommt
6. Resultierenden Dataframe sortieren nach true und false -> in Verbindung mit mutcount bringen
7. innerhalb einer Mutationsspalte, alle mit dem selben mutcount: fscore vergleich : Varianz der Werte als Maß für impact

- -> je unwichtiger die Mutation, desto weniger macht sie was beim fscore, desto weniger sollten sich die Werte unterscheiden, desto kleiner sollte die Varianz sein

In [16]:

```
def checking_existence(soll_gecheckt_werden, Datenset):
    list_of_dfs = []

    for i in soll_gecheckt_werden:
        new_column_name = f'{i}'
        new_column_values = [ i in x for x in Datenset]
        new_df = pd.DataFrame({new_column_name: new_column_values})
        list_of_dfs.append(new_df)

    result_how_often = pd.concat(list_of_dfs, axis=1)
    print(result_how_often)

checking_existence(all_possible_mutations, only_mutants_list)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 13
     10     result_how_often = pd.concat(list_of_dfs, axis=1)
     11     print(result how often)
---> 13 checking_existence(all_possible_mutations, only_mutants_list)

Cell In[16], line 6, in checking_existence(soll_gecheckt_werden, Datenset)
      4 for i in soll_gecheckt_werden:
      5     new_column_name = f'{i}'
----> 6     new_column_values = [ i in x for x in Datenset]
      7     new_df = pd.DataFrame({new_column_name: new_column_values})
      8     list_of_dfs.append(new_df)

Cell In[16], line 6, in <listcomp>(.0)
      4 for i in soll_gecheckt_werden:
      5     new_column_name = f'{i}'
----> 6     new_column_values = [ i in x for x in Datenset]
      7     new_df = pd.DataFrame({new_column_name: new_column_values})
      8     list_of_dfs.append(new_df)

TypeError: 'in <string>' requires string as left operand, not NoneType
```

Problem 1: egal wie ich versuche das rauszufinden, denkt er in der Liste wären „none“ Werte enthalten
Getestet: in keiner der beiden Listen sind welche ??

Datei: L_second_try_stabilizing_mutations.ipynb

LISA 5: ANFANG STABILISIERENDE MUTATIONEN

Problem 2: Ist es normal das deer Code mehrere Stunden läuft? Pycharm die ganze zeit halb am abstürzen?
-> Alternativen zu for loops oder apply() Funktionen?

- Plan:

1. Angelas Code in meinen übersetzen
2. aus Angelas Code alle möglichen Mutationen rausschreiben, aber nur einmal
3. --> Liste mit allen vorkommenden Mutationen
4. Funktion schreiben, die für jede dieser (einzeln!) alle Sequenzen raussucht in der diese vorkommt
5. --> ein dataframe, mit mutation als spaltenname und boolians als Angabe ob sie in der jeweiligen Mutante(Zeile) vorkommt

Next:

6. Resultierenden Dataframe sortieren nach true und false -> in Verbindung mit mutcount bringen
7. innerhalb einer Mutationsspalte, alle mit dem selben mutcount: fscore vergleich : Varianz der Werte als Maß für impact

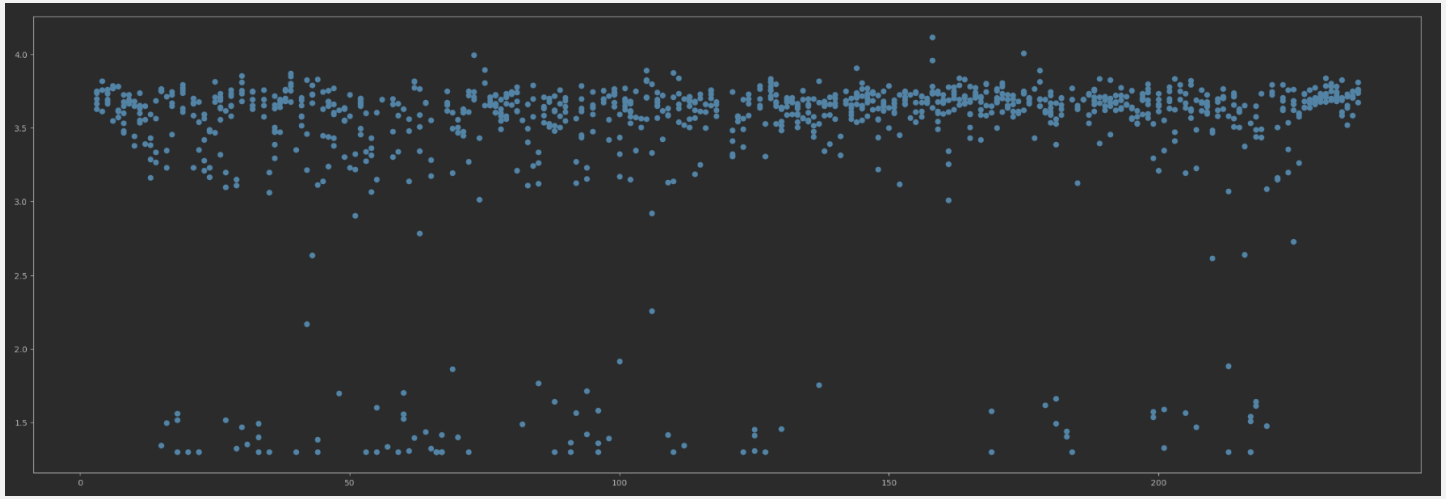
- -> je unwichtiger die Mutation, desto weniger macht sie was beim fscore, desto weniger sollten sich die Werte unterscheiden, desto kleiner sollte die Varianz sein

ANGELA I

(file: A_fscore_position)

- mutationsintolerante Positionen✅

- Wie ist eine bessere Achsenbeschriftung mögl.? (gleicht sich mit Romans Frage)
  - Skalierung in 1-er-Schritten
  - und gleichzeitig keine Überlappung d. Zahlen

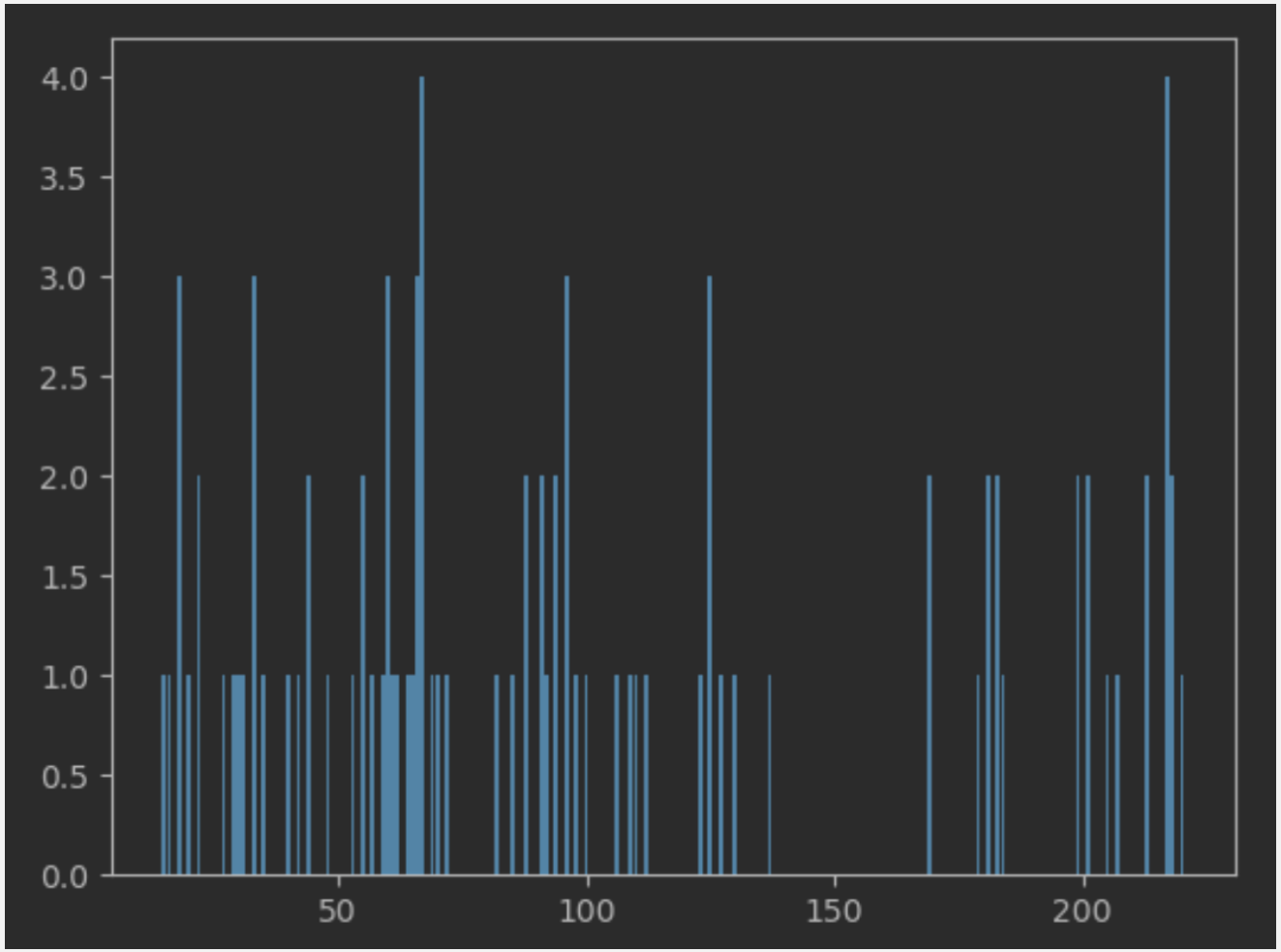


ANGELA 2

(file: A_fscore_position)

- Anzahl Mutanten mit single mutation, wobei fscore < 2,5

- Problem: wenig Mutanten mit single mutations → man kann nur wage Aussagen machen
- bestätigt aber u.a. nochmal, dass Chromophor essenziel ist (mutationsintolerant)



ANGELA 3

(file: A_Versuch aufeinanderfolgende Mutationen rausfiltern)

- fscore v. Mutanten mit teilweise gleichen Mutationen vergleichen

- Ansatz
  1. Zeilen mit unique values in ´m1´ löschen
  2. merge ´m1´ and ´m2´ → ´m1m2´
  3. keep rows with identical values in ´m1m2´
  4. Step 3 nochmal f. ´m1-m3´wiederholen

```
# only keep mutants with same m1
df_without_unique_m1 = df_only_fscore_mutations

df_without_unique_m1 = df_without_unique_m1[df_without_unique_m1['m1'].map(df_without_unique_m1['m1'].value_counts
 ()) > 1]

df_without_unique_m1.head()
print(df_without_unique_m1)
```

Executed at 2023.05.30 21:48:42 in 82ms

	DMS_score	m1	m2	m3	m4	m5	m6	m7	m8	m9
0	1.301030	K3R	V55A	Q94R	A110T	D117G	M153K	D216A	None	None
1	3.137350	K3Q	V16A	I167T	L195Q	None	None	None	None	None
2	1.553913	K3Q	Y143C	N164D	S205P	A227T	None	None	None	None
3	3.404237	K3Q	Y143N	V193A	None	None	None	None	None	None
4	3.738586	K3R	None	None	None	None	None	None	None	None
...	...	...	...	...	...	...	...	...	...	...
51694	3.865135	D234V	Y237H	None	None	None	None	None	None	None
51696	3.693903	D234G	E235V	None	None	None	None	None	None	None
51697	3.706313	D234G	None	None	None	None	None	None	None	None
51699	3.862731	E235G	L236P	None	None	None	None	None	None	None
51700	3.727375	E235G	None	None	None	None	None	None	None	None

```
# keep rows with identical values in 'm1m2'
# problem: rows with single mutations deleted
df_identical_m1m2 = df_merge_m1_m2

df_identical_m1m2 = df_identical_m1m2[df_identical_m1m2.duplicated(subset=['m1m2'], keep=False)]

df_identical_m1m2.head()
print(df_identical_m1m2)
```

Executed at 2023.05.30 22:09:12 in 9ms

	DMS_score	m1	m2	m3	m4	m5	m6	m7	m8	m9
0	1.301030	K3R	V55A	Q94R	A110T	D117G	M153K	D216A	None	None
5	3.851893	K3R	A87T	D173G	None	None	None	None	None	None
6	3.551648	K3R	A87T	N144S	T225S	None	None	None	None	None
8	3.388889	K3R	D102G	N185D	L195P	H231P	E235K	None	None	None
9	3.499501	K3R	D102G	Y151C	N170D	I229T	None	None	None	None
...	...	...	...	...	...	...	...	...	...	...

ANGELA 4

(file: A_Versuch aufeinanderfolgende Mutationen rausfiltern)

- Ergebnis: 113 Zeilen mit gleichen Mutationen m1 bis m3

- aber: kein einziges Mutantenpaar, die von m1 bis m4 übereinstimmen

- Schwachpunkt: m1 wird als Ausgangspunkt genommen
  - → Mutanten, die sich nicht in ´m1´, aber in anderen Spalten / spaltenübergreifend gleichen, werden auch gelöscht (unerwünscht)
- → neuer Ansatz
  - überlappende Mutationen spaltenübergreifend checken

ANGELA 5

(file: A_Stammbaum_across_m1_to_m15)

- for loop, um spaltenübergreifend zu checken

- Probleme
  - dauert mehr als 2 Stunden
  - alle Zeilen erfüllen die Bedingung
    - entweder: alle Mutanten überlappen sich mit mind. 1 anderen Mutante?

- nochmal für "2 mutations in common"?

```python
# keep mutants that have at least 1 mutation in common
df_one_in_common = df_only_fscore_mutations

# for loop for checking mutants with at least 1 mutation in common
# check for every row
for i, row in df_one_in_common.iterrows():
    # check mutations that are in common
    if any(set(row.values).intersection(set(x.values)) for _, x in df_one_in_common.drop(i).iterrows()):
            df_one_in_common = df_one_in_common.append(row)
```

Executed at 2023.05.31 11:17:47 in 2h 8m 32s

```
[51714 rows x 19 columns]
```

```
[103428 rows x 16 columns]
```

ANGELA 6

Next Steps:

- Probleme aus Folien 3 bis 5 lösen
- wenn gelöst:
  - → Trend fitness score?
  - → Verbesserung nach Verschlechterung (u. vice versa)?
- plots f. Ergebnisse oben
- mit Lisas Codes kombinieren und damit mögliche Analysen diskutieren