analyse

February 15, 2025

[29]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import copy
from Bio import SeqIO
```

1 Outils

- **notebook** : un type de fichier permettant d'intercaler des section de code et des section de texte, permet aussi d'afficher les plots directements (pratique pour décrire une analyse)
- **pyvcf** : librairie python pour parser des fichier vcf et récupérer les variants qui y sont référencé
- **Biopython** : librairie de bioinformatique qui nous permet ici de parser le fichier fasta contenant toutes les ORF du virus
- **pandas** : Permet de traiter des donnés tabulaires (type csv) efficacement et facilement (accession, filtrage, trie, …) dans un objet appelé data_nf1Frame
- **seaborn** : Permet de réaliser des graphiques facilement à partir de DataFrame

2 Récupération des données

Les variants structuraux de tous les échantillons pour tous les passage ont été récupérés au préalable sur le cluster et regroupés dans un fichier csv. Chaque variants est représenté par une donné avec les entrées suivantes: - **id** : identifiant de la variation dans le fichier vcf - **svtype** : type de variant (INS, DEL, INV, DUP) - **pos** : position de début de la variation - **end** : position de fin de la variation (pour une insertion pos = end) - **svlen** : taille de la variation (négatif pour les délétions) - **alt** : séquence du variant en cas d'insertion - **dr, dv** : profondeur de read mappant sur la référence (dr) et mappant sur le variant (dv) - **depth** : profondeur de read total sur cette région du génome (dr + dv) - **af** : fréquence allélique (dv / depth : proportion de read supportant cette inversion par rapport au reads mappés sur cette région) - **sample** : échantillon d'origine (1 à 10) - **iteration** : passage d'origine (15 à 90) - **group** : les variants identiques sont regroupés avec un identifiant identique (voir ci dessous)

2.1 Groupement des variants identiques

On veut pouvoir repérer les différentes occurences d'un même variant dans différentes expériences. Pour cela les variants sont comparé entre eux en se basant sur les éléments suivants, qui doivent

être identiques : - La position de début - La position de fin - La séquence alternative pour les insertions

Il a été choisis de grouper les variants seulement lorsqu'ils sont exactement identique pour être certain qu'ils ont le même impacte sur le fonctionnement biologique. Par ailleurs, cette méthode identifie de nombreuses occurences d'un même variant, ce qui montre que ce seuil est pertinent : pour 2 457 variants, 779 groupes ont été identifiés

[30]:
```
data = pd.read_csv('../data/variants.csv', index_col="index")
data["choc"] = data.apply(lambda x: "cold" if x["sample"] <= 5 else "heat",␣
 ↪axis=1)

data["iteration"] = pd.Categorical(data["iteration"])
data["sample"] = pd.Categorical(data["sample"])
data["group"] = pd.Categorical(data["group"])
data["svtype"] = pd.Categorical(data["svtype"])
data["choc"] = pd.Categorical(data["choc"])

data.head()
```

[30]:

index	pos	id	svtype	svlen	end	af	dv	dr	depth	alt	sample	iteration	group	choc
0	1	Sniffles2.DUP.705S0	DUP	272677	272678	0.0	0	0	0	NaN	1	15	0	cold
1	1	Sniffles2.DUP.3B3S0	DUP	272677	272678	0.0	0	0	0	NaN	1	30	0	cold
2	1	Sniffles2.DUP.29FS0	DUP	272677	272678	0.0	0	0	0	NaN	1	50	0	cold
3	1	Sniffles2.DUP.6ADS0	DUP	272677	272678	0.0	0	0	0	NaN	2	15	0	cold
4	1	Sniffles2.DUP.1AS0	DUP	272677	272678	0.0	0	0	0	NaN	2	30	0	cold

[31]:
```
data_init = pd.DataFrame(
    columns = data.columns, data = copy.deepcopy(data.values)
)
data_init["iteration"] = pd.Categorical(data["iteration"])
data_init["sample"] = pd.Categorical(data["sample"])
data_init["group"] = pd.Categorical(data["group"])
data_init["svtype"] = pd.Categorical(data["svtype"])
```

3 Filtrage des données

3.1 Fréquence allélique et profondeur

Pour déterminer des filtres de fréquences alléliques et de profondeur minimum qui soit pertinent on regarde quelles sont les valeurs observés dans les différents passage. Il est nécessaire de regarder tous les passages indépendament puisque la qualité globale du séquencage diffère d'un passage à l'autre. On choisiras cependant un seuil fixe pour l'ensemble des passage afin de rester consistant.

Pour les fréquences allélique, on voit sur l'histogramme on remarque le "coude" de la distribution à 0.05 (ligne rouge) : comme une grande quantité d'observations se situe en dessous de ce seuil, il est pertinent de considérer que ces observations sont les moins significatives.

Ce raisonnement est plus compliqué à appliquer sur les profondeur qui ne sont pas aussi distinctement répartis, on peut afficher des violin plots pour voir la distribution de chaque passage. La largeur du graphique représente la proportion de reads à une valeur donnée de profondeur, un boxplot est représenté à l'interieur en noir. On voit qu'avec un filtre à 1100 on écarte moins de 25% des variants, cette valeur semble pertinente.

Le dernier graphique représente chaque variant par un point positioné en fonction de sa profondeur (absysses) sa fréquence allélique (ordonnées).

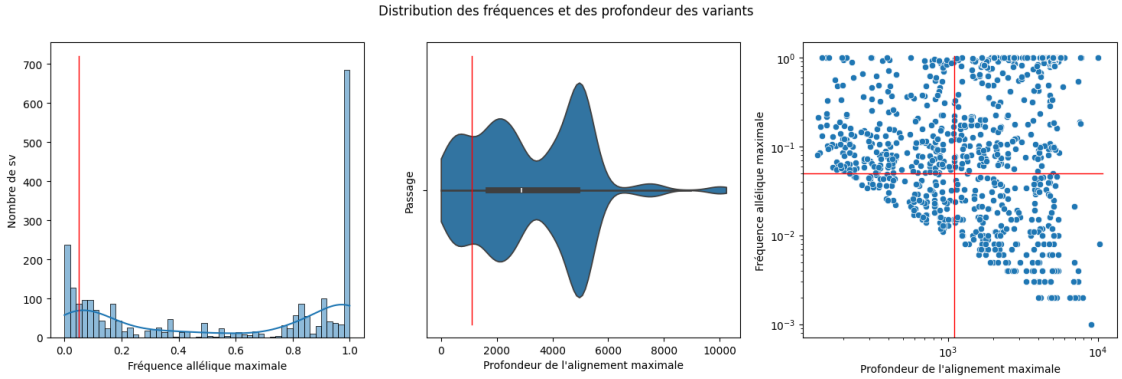
Enfin avec des seuils à 0.05 et 1100 on conservera des variants supportés par 55 reads minimum ce qui est significatif (0.05 * 1100).

Remarque : ce sont les groupes de variants (cf description plus haut) identiques qui sont filtrés et non les variants individuellement, par exemple si un variant est présent à P15 et ne remplis pas les conditions pour passer le filtre, mais qu'il est aussi présent à P50, cette fois en fréquence et en profondeur suffisante, les deux occurences seront conservé. En effet cela nous donne un information plus pertinentes sur les variants et nous permets de considérer les différents passages ensembles. On affiche donc les fréquences maximale pour chaque groupe de variants identiques.

[32]:
```
grouped =  data_init.groupby(["group"], observed=True)
data_init["max_af"] = data_init.apply(lambda x: grouped.
 ↪get_group(x["group"])["af"].max(), axis=1)
data_init["max_depth"] = data_init.apply(lambda x: grouped.
 ↪get_group(x["group"])["depth"].max(), axis=1)
```

[33]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))
sns.histplot(data=data_init, x="max_af", kde=True, bins=50, ax=ax1)
sns.violinplot(data=data_init, x="max_depth", density_norm="area", ax=ax2,␣
 ↪cut=0)
sns.scatterplot(data=data_init[(data_init["iteration"].isin([90, 15, 65, 30,␣
 ↪50])) & (data_init["depth"] > 50)], x="max_depth", y="max_af", ax=ax3)

ax1.plot([0.05, 0.05], [0.0, ax1.get_ylim()[1]], 'r-', lw=1)
ax1.set_xlabel("Fréquence allélique maximale")
ax1.set_ylabel("Nombre de sv")

ax2.plot([1100.0, 1100.0], [ax2.get_ylim()[0], ax2.get_ylim()[1]], 'r-', lw=1)
```

```
ax2.set_xlabel("Profondeur de l'alignement maximale")
ax2.set_ylabel("Passage")

ax3.plot([ax3.get_xlim()[0], ax3.get_xlim()[1]], [0.05, 0.05], 'r-', lw=1)
ax3.plot([1100.0, 1100.0], [ax3.get_ylim()[0], ax3.get_ylim()[1]], 'r-', lw=1)
ax3.set_xlabel("Profondeur de l'alignement maximale")
ax3.set_ylabel("Fréquence allélique maximale")
ax3.set_yscale("log")
ax3.set_xscale("log")

plt.suptitle("Distribution des fréquences et des profondeur des variants")
plt.show()
```



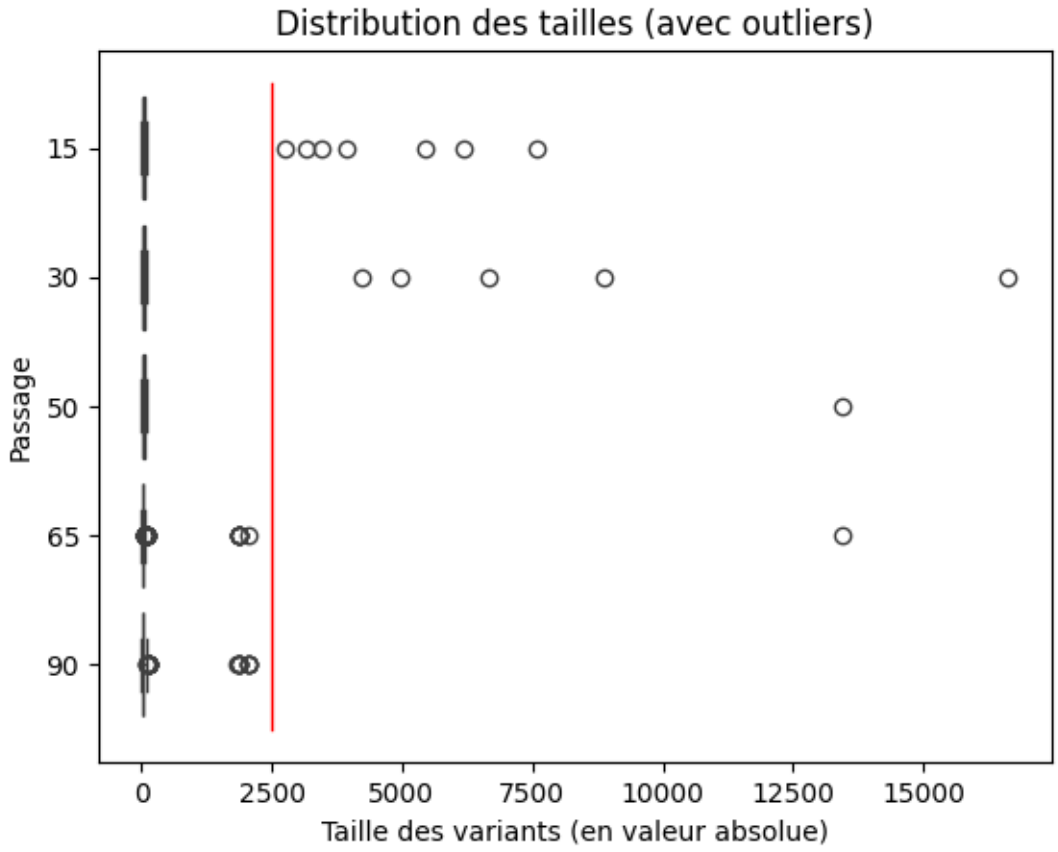
[34]:
```
filter_af = 0.05
filter_depth = 1100
data = data.groupby(["group"], observed=True).filter(lambda x: x["af"].max() >␣
↪filter_af and x["depth"].max() > filter_depth)
```

3.2 Repérer les outliers

Des outliers sont des données inattendu qui peuvent possiblement être causé par des bugs. Dans notre cas on sait que les régions au début et à la fin du génome sont répétés, ce qui peut conduire à détecter de larges insertions dans ces régions. Afficher la distribution des longueurs de variants dans un box plot nous permet de les repérer pour les filtrer correctement.

[35]:
```
fig, ax = plt.subplots()
data_filt = data_init.groupby(["group"], observed=True).filter(lambda x:␣
↪x["af"].max() > filter_af and x["depth"].max() > filter_depth)
sns.boxplot(data=data_filt[(30000 < data_filt["pos"]) & (data_filt["pos"] <␣
↪270000)], x=abs(data[(30000 < data_filt["pos"]) & (data_filt["pos"] <␣
↪270000)]["svlen"]), y="iteration", showfliers=True, ax=ax)
ax.set_xlabel("Taille des variants (en valeur absolue)")
ax.set_ylabel("Passage")
```

```
ax.set_title("Distribution des tailles (avec outliers)")

plt.plot([2500.0, 2500.0], [ax.get_ylim()[0], ax.get_ylim()[1]], 'r-', lw=1)
plt.show()
```



En observant plus en détails ces donnés on remarque que soit ces variants sont positionnés au bornes du génomes, soit ce sont des régions répétés. Ci dessous tous les variants d'une taille superieur à 2 500 entre les positions 20 000 et 280 000.

On décidera donc de filtrer les variants qui ont une taille supérieur à 2 500

[36]:
```
def truncate(x):
    if x["svtype"] == "INS":
        x["alt"] = x["alt"][max(len(x["alt"]) - 20, 0):]
    return x

outliers = data.copy(deep=True)
outliers["svlen"] = abs(outliers["svlen"])
outliers = outliers[(outliers["svlen"] > 2500) & (outliers["pos"] > 20000) &␣
 ↪(outliers["pos"] < 280000)]
```

```
outliers = outliers.sort_values("pos").sort_values("svlen", ascending=False)

outliers = outliers.apply(truncate, 1)
outliers[["svlen", "pos", "alt"]]
```

[36]:

index	svlen	pos	alt
1774	16622	196282	AAAAAAAAAAAAAAAAAAAA
2018	13429	229512	NaN
2019	13429	229512	NaN
1333	8855	138818	CTCAACTGTCGTGCCGTTGA
1082	7568	85419	GGGGGGGGGGCGCTATGTGG
1447	6668	155479	CTCTGGGGCTGAGCGGGAGA
1334	6168	142608	GGGGGGGGGGGGGGGGGGGG
303	5867	22914	GAGAGAGAGAGAGAGAGAGA
1267	5459	122251	GGGGGGGGGGGGGGGGGGGG
848	4977	54238	GGGGGGGGGGGGGGGGGGGG
2203	4226	254597	AGGGAGAGCGCGCGCGCGCG
1947	3921	225969	GTGTGTGTGTGTGCGGCGGT
1930	3453	217321	GGGGGGGGGGGGGGGGGGGG
967	3157	81347	CGGTCAAGAGCTGAGACTGC
2281	2758	264795	AGCGAGGAGGAGGAGGAGAG

[37]:
```
data = data[abs(data["svlen"]) < 2500]
```

3.3 Impacte du filtrage

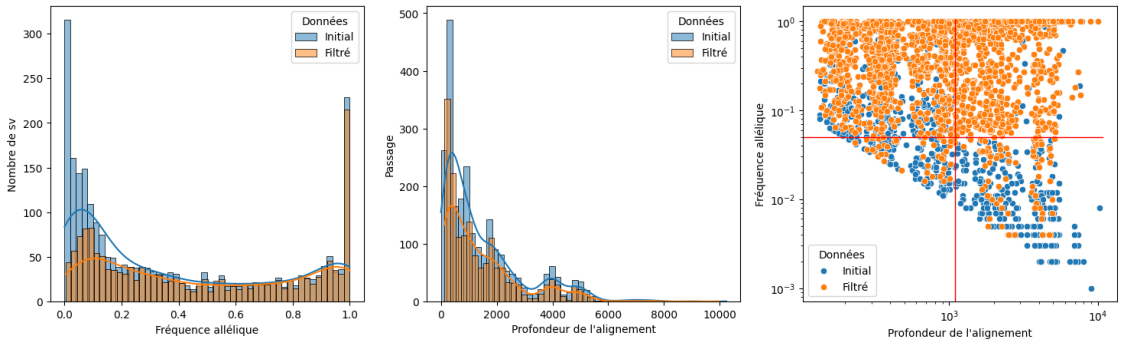
On superpose les données aux différentes étapes du filtrage pour comprendre l'impacte et voir la quantité de données conservés.

Ici j'ai choisit de montrer toutes les fréquences indépendament. On voit que même avec un filtre élevé concernant la profondeur on conserve une grande proportion des variants.

[38]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.histplot(data=data_init, x="af", label="Initial", kde=True, bins=50, ax=ax1)
sns.histplot(data=data, x="af", label="Filtré", kde=True, bins=50, ax=ax1)
ax1.legend(title="Données")

sns.histplot(data=data_init, x="depth", label="Initial", kde=True, bins=50,␣
↪ax=ax2)
sns.histplot(data=data, x="depth", label="Filtré", kde=True, bins=50, ax=ax2)
ax2.legend(title="Données")

sns.scatterplot(data=data_init[(data_init["depth"] > 50)], x="depth", y="af",␣
↪label="Initial", ax=ax3)
sns.scatterplot(data=data[(data["depth"] > 50)], x="depth", y="af",␣
↪label="Filtré", ax=ax3)
```

```
ax3.legend(title="Données")

ax1.set_xlabel("Fréquence allélique")
ax1.set_ylabel("Nombre de sv")

ax2.set_xlabel("Profondeur de l'alignement")
ax2.set_ylabel("Passage")

ax3.plot([ax3.get_xlim()[0], ax3.get_xlim()[1]], [filter_af, filter_af], 'r-',␣
 ↪lw=1)
ax3.plot([filter_depth, filter_depth], [ax3.get_ylim()[0], ax3.get_ylim()[1]],␣
 ↪'r-', lw=1)
ax3.set_xlabel("Profondeur de l'alignement")
ax3.set_ylabel("Fréquence allélique")
ax3.set_yscale("log")
ax3.set_xscale("log")

print("Proportion de variants conservés :", len(data)/len(data_init))
plt.show()
```

```
Proportion de variants conservés : 0.6919006919006919
```



3.4 Extraire les variants qui interfèrent avec un ORF connu

Pour cela on extrait tous les o connues du virues à partir d'un fichier FASTA. Si une variation structurelle chevauche un ou plusieurs o, on noteras lesquel.

[39]:
```
orfs = []

for seq in SeqIO.parse("../ORF.fasta", "fasta"):
    header = seq.description

    locus_tag = header.split("[")[1][:-2].split("=")[1]
    protein_id = header.split("[")[4][:-2].split("=")[1]
```

```
    locations = header.split("[")[5][:-2].split("(")[-1].split("=")[-1].
↪split(")")[0].split(",")
    locations = [inter.split("..") for inter in locations]

    complement = 'complement' in header.split("[")[5]
    join = 'join' in header.split("[")[5]

    to_add = [{'locus_tag': locus_tag, 'protein_id': protein_id, 'location':␣
↪loc, 'complement': complement, 'join': join} for loc in locations]
    orfs += to_add


def intersected_orfs(pos, end):
    result = []
    for o in orfs:
        if min(end, int(o["location"][1])) - max(pos, int(o["location"][0])) >␣
↪0:
            result.append(o["locus_tag"])
    return result
```

[40]:
```
data["orfs"] = data.apply(lambda x: intersected_orfs(x["pos"], x["end"]),␣
↪axis=1)
data[data["orfs"].astype(bool)][["id", "orfs"]].head()
```

[40]:

index	id	orfs
54	Sniffles2.DEL.1D5S0	[CyHV3_ORF5_1]
668	Sniffles2.DEL.1E6S0	[CyHV3_ORF25, CyHV3_ORF26, CyHV3_ORF27]
669	Sniffles2.DEL.34BS0	[CyHV3_ORF25, CyHV3_ORF26, CyHV3_ORF27]
670	Sniffles2.DEL.2C4S0	[CyHV3_ORF25, CyHV3_ORF26, CyHV3_ORF27]
671	Sniffles2.DEL.263S0	[CyHV3_ORF25, CyHV3_ORF26, CyHV3_ORF27]

4 Exploration de P90

Remarque : Le code suivant peut être appliqué aux autres passages simplement en changant la variale ioi

[41]:
```
ioi = 90 # Iteration of interests

data_oi = data[data["iteration"] == ioi]
data_oi_orf = data_oi[data_oi["orfs"].astype(bool)]
```

4.1 Distributions des variants

On veut d'abord mener une analyse descriptive pour voir les différents variants qui compose les échantillons du passage P90, leur quantités, leurs positions et leur tailles.

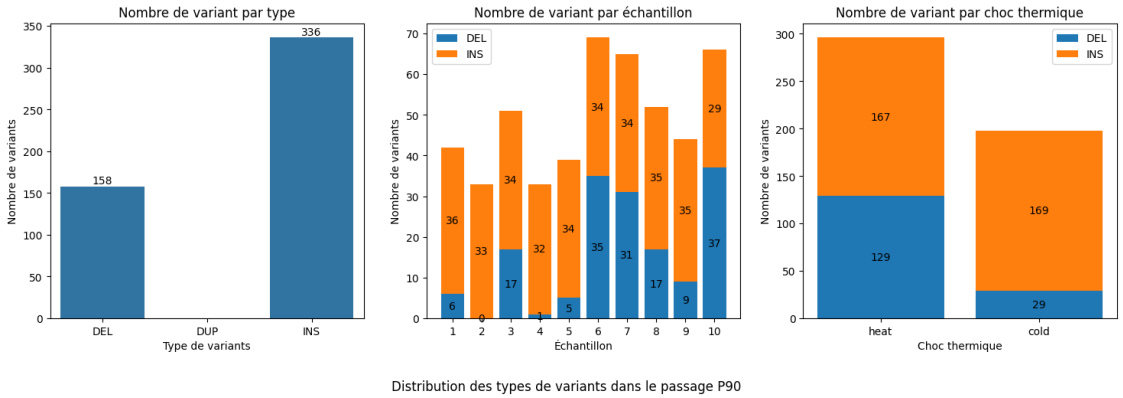
On remarque que les échalntillons du choc chaud sont beaucoup plus riche en délétion (3ème plot)

alors que le nombre d'insertions reste très stable entre les différents échantillons (2ème plot). Le nombre d'insertion étonnament proche entre le groupe chaud et froid nous laisse penser que ce pourrait être les mêmes insertions. À ce stade on peut supposer que les délétions sont plus corrélés au choc thermique que les insertions, elles ont pu être sélectionnés dasn le chaud ou éliminés dans le froid.

Si on suppose que la probabilité d'apparition d'une insertion et d'une délétions sont identiques (est ce que c'est vrai ?), on pourrait alors imaginer que les délétions sont contre-sélectionnés par le choc froid, donc ont un effet délétaire pour ce groupe.

[42]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.countplot(data=data_oi, x="svtype", ax=ax1)
ax1.bar_label(container = ax1.containers[0], fontsize=10)
ax1.set_ylabel('Nombre de variants')
ax1.set_xlabel('Type de variants')
ax1.set_title(f"Nombre de variant par type")

type_count_sample = data_oi.groupby(["svtype", "sample"], observed=True).size()
types = data_oi["svtype"].unique()
samples = data_oi["sample"].unique()
sorted(samples)
bottom = np.zeros(len(samples))
for t in types:
    gc = [type_count_sample[t][s] if s in type_count_sample[t] else 0 for s in␣
 ↪samples]
    p = ax2.bar(samples, gc, label=t, bottom=bottom)
    bottom += gc
    ax2.bar_label(p, label_type='center')

ax2.set_ylabel('Nombre de variants')
ax2.set_xlabel('Échantillon')
ax2.set_xticks(samples)
ax2.set_title(f"Nombre de variant par échantillon")
ax2.legend()


type_count_choc = data_oi.groupby(["svtype", "choc"], observed=True).size()
types = data_oi["svtype"].unique()
chocs = data_oi["choc"].unique()
bottom = np.zeros(len(chocs))
for t in types:
    gc = [type_count_choc[t][c] if c in type_count_choc[t] else 0 for c in␣
 ↪chocs]
    p = ax3.bar(chocs, gc, label=t, bottom=bottom)
    bottom += gc
    ax3.bar_label(p, label_type='center')
```

```
ax3.set_ylabel('Nombre de variants')
ax3.set_xlabel('Choc thermique')
ax3.set_title(f"Nombre de variant par choc thermique")
ax3.legend()

plt.suptitle(f"Distribution des types de variants dans le passage P{ioi}", y=-0.
↪05)
plt.show()
```

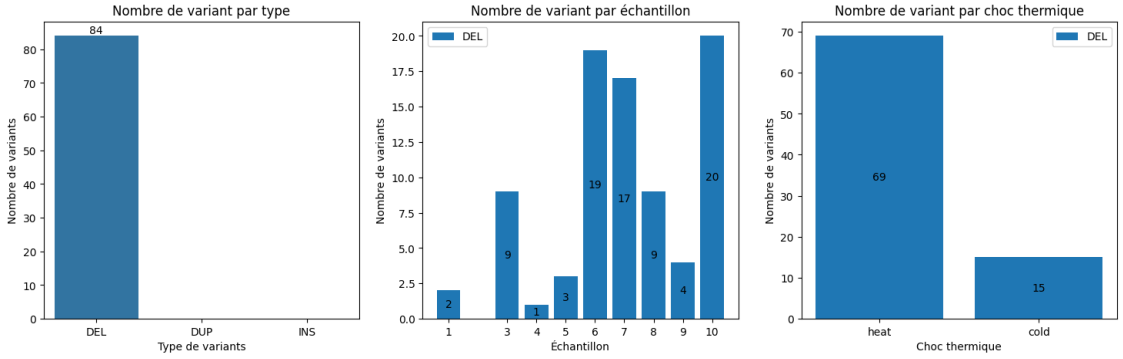


On peut répliquer l'expérience en s'intéressant uniquement aux variants présent dans des ORFs.

Les insertions qui sont présente n'ont a priori pas d'impacte sur les ORF, ce qui nous comforte dans l'hypothèse que c'est pour les délétions que le choc thermique à été discriminant.

[43]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.countplot(data=data_oi_orf, x="svtype", ax=ax1)
ax1.bar_label(container = ax1.containers[0], fontsize=10)
ax1.set_ylabel('Nombre de variants')
ax1.set_xlabel('Type de variants')
ax1.set_title(f"Nombre de variant par type")

type_count_sample = data_oi_orf.groupby(["svtype", "sample"], observed=True).
↪size()
types = data_oi_orf["svtype"].unique()
samples = data_oi_orf["sample"].unique()
sorted(samples)
bottom = np.zeros(len(samples))
for t in types:
    gc = [type_count_sample[t][s] if s in type_count_sample[t] else 0 for s in␣
↪samples]
    p = ax2.bar(samples, gc, label=t, bottom=bottom)
    bottom += gc
```

```
    ax2.bar_label(p, label_type='center')

ax2.set_ylabel('Nombre de variants')
ax2.set_xlabel('Échantillon')
ax2.set_xticks(samples)
ax2.set_title(f"Nombre de variant par échantillon")
ax2.legend()


type_count_choc = data_oi_orf.groupby(["svtype", "choc"], observed=True).size()
types = data_oi_orf["svtype"].unique()
chocs = data_oi_orf["choc"].unique()
bottom = np.zeros(len(chocs))
for t in types:
    gc = [type_count_choc[t][c] if c in type_count_choc[t] else 0 for c in␣
 ↪chocs]
    p = ax3.bar(chocs, gc, label=t, bottom=bottom)
    bottom += gc
    ax3.bar_label(p, label_type='center')

ax3.set_ylabel('Nombre de variants')
ax3.set_xlabel('Choc thermique')
ax3.set_title(f"Nombre de variant par choc thermique")
ax3.legend()

plt.suptitle(f"Distribution des types de variants dans le passage P{ioi}␣
 ↪(variants dans les ORFs uniquement)", y=-0.05)
plt.show()
```



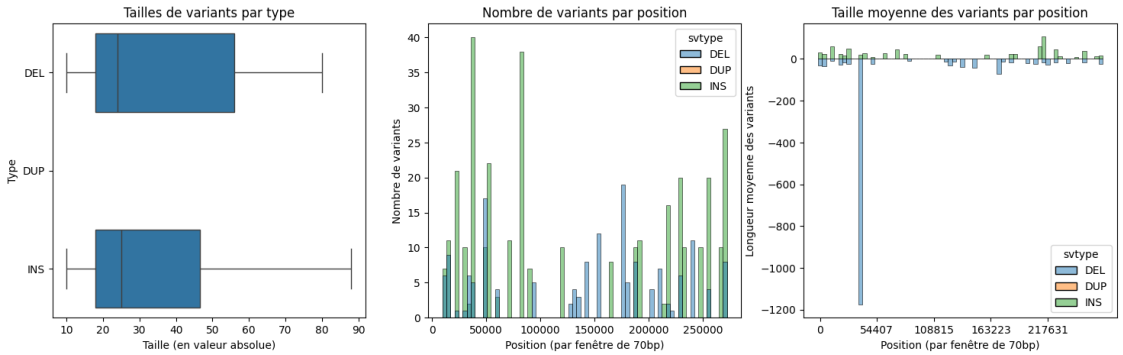
Distribution des types de variants dans le passage P90 (variants dans les ORFs uniquement)

4.2 Tailles et positions des variants

On veut analyser la taille des différents variants ainsi que leurs position sur le génome pour voir si il y a des régions qui se démarquent.

On note qu'il n'y a pas de différence notable de longueur entre les insertions et les délétions, ormis la délétion "géante" de l'orf25

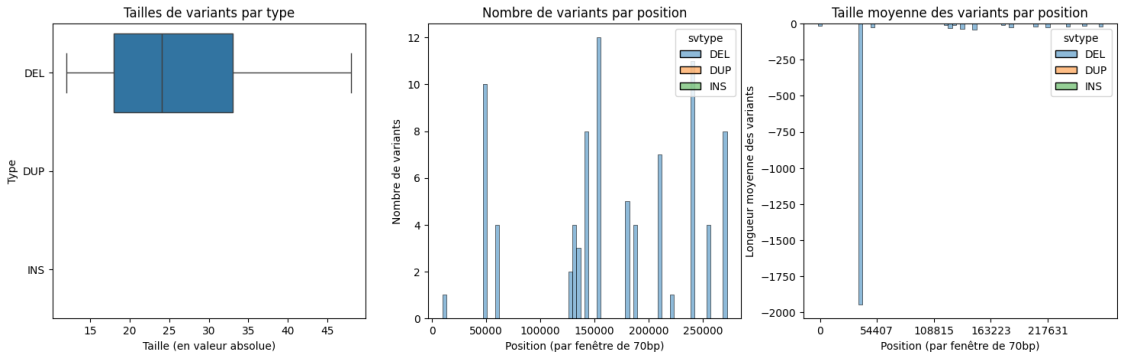
[44]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.boxplot(data=data_oi, x=abs(data_oi["svlen"]), y="svtype",␣
 ↪showfliers=False, ax=ax1)
ax1.set_xlabel("Taille (en valeur absolue)")
ax1.set_ylabel("Type")
ax1.set_title("Tailles de variants par type")

nbins = 70
sns.histplot(data=data_oi, x="pos", hue="svtype", bins=nbins, ax=ax2)
ax2.set_xlabel(f"Position (par fenêtre de {nbins}bp)")
ax2.set_ylabel("Nombre de variants")
ax2.set_title("Nombre de variants par position")

max_pos = max(data_oi["pos"])
data_oi.loc[:, 'pos_bin'] = pd.cut(data_oi['pos'], nbins, labels=range(nbins))
data_mean = data_oi.groupby(['pos_bin', 'svtype'], as_index=False,␣
 ↪observed=True)['svlen'].mean()
sns.histplot(data=data_mean, x="pos_bin", hue="svtype", weights='svlen',␣
 ↪bins=nbins, ax=ax3)
ax3.set_xlabel(f"Position (par fenêtre de {nbins}bp)")
ax3.set_ylabel("Longueur moyenne des variants")
ax3.set_xticks([i * (nbins / 5) for i in range(5)], labels=[int(i * (max_pos /␣
 ↪5)) for i in range(5)])
ax3.set_title("Taille moyenne des variants par position")

plt.suptitle(f"Distribution des tailles et positions des variants dans le␣
 ↪passage P{ioi}", y=-0.05)
plt.show()
```



Distribution des tailles et positions des variants dans le passage P90

Similairement, on peut répliquer ces graphiques sur les variants présent uniquement dans les ORFs

[45]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.boxplot(data=data_oi_orf, x=abs(data_oi_orf["svlen"]), y="svtype",␣
 ↪showfliers=False, ax=ax1)
ax1.set_xlabel("Taille (en valeur absolue)")
ax1.set_ylabel("Type")
ax1.set_title("Tailles de variants par type")

nbins = 70
sns.histplot(data=data_oi_orf, x="pos", hue="svtype", bins=nbins, ax=ax2)
ax2.set_xlabel(f"Position (par fenêtre de {nbins}bp)")
ax2.set_ylabel("Nombre de variants")
ax2.set_title("Nombre de variants par position")

max_pos = max(data_oi["pos"])
data_oi_orf.loc[:, 'pos_bin'] = pd.cut(data_oi_orf['pos'], nbins,␣
 ↪labels=range(nbins))
data_mean = data_oi_orf.groupby(['pos_bin', 'svtype'], as_index=False,␣
 ↪observed=True)['svlen'].mean()
sns.histplot(data=data_mean, x="pos_bin", hue="svtype", weights='svlen',␣
 ↪bins=nbins, ax=ax3)
ax3.set_xlabel(f"Position (par fenêtre de {nbins}bp)")
ax3.set_ylabel("Longueur moyenne des variants")
ax3.set_xticks([i * (nbins / 5) for i in range(5)], labels=[int(i * (max_pos /␣
 ↪5)) for i in range(5)])
ax3.set_title("Taille moyenne des variants par position")

plt.suptitle(f"Distribution des tailles et positions des variants dans le␣
 ↪passage P{ioi} (dans les ORFs uniquement)", y=-0.05)
plt.show()
```



Distribution des tailles et positions des variants dans le passage P90 (dans les ORFs uniquement)

4.3 Comparaisons entre les échantillons

4.3.1 Méthode

On évalue la similarité entre échlantillons, paire à paire, pour voir si ceux ci partagent beaucoup de variations. On sait déjà des analyse précédentes que le groupe chaud possède nettement plus de délétions que le groupe froid. Cette analyse nous permet de voir si ces délétions sont uniques, ou si elles sont partagés entre les échantillons du groupe chaud : si on les retrouves en quantité significatives, nous pourrons affirmer avec un certain niveau de confiance que la présence/absence de ces mutations est belle est bien corrélé au choc thermique.

Soit S la matrice de similarité, la similarité entre deux échantillons i, j est donné par $S_{i,j} = Sj, i = \frac{\text{Nombre de variants présent dans } \mathbf{i \text{ et } j}}{\text{Nombre de variants présents dans } \mathbf{i \text{ ou } j}}$

Bien que cette mesure donne une bonne idée de l'homogénéité des échantillons, il pourrait être intéressant de considérer un vrai test statistique qui serait pet être plus robuste. La principale faiblesse ici est la sensibilité à la taille des échantillons : en effet si les échantillons sont trop petits, ils sont nécéssairement moins suceptible de partager des variations, d'autant que les échantillons du groupe froid présentent moins de variations que les échantillons du groupe chaud (voir plus haut).

4.3.2 Résultat

Dans la 1ère matrice on observe des valeurs plus significative au sein des groupes froid / chaud, la tendance est assez similaire. Bien que les insertions présentent dans le groupe froid ne se trouvent pas dans des ORFs, l'homogénéité entre les échantillons 1 à 5 peut nous laisser penser que: - ces insertions ont un impacte sur la fitness du virus (région régulatrice ? apparition d'un ORF ? quoi d'autre ?) -> comment ont elles évolué depuis les précédents passages - ces insertions se trouvent dans des régions fortement mutagènes -> investiguer les régions de ces insertions - ces insertions sont des artéfact causés par des régions répétés -> investiguer les régions de ces insertions

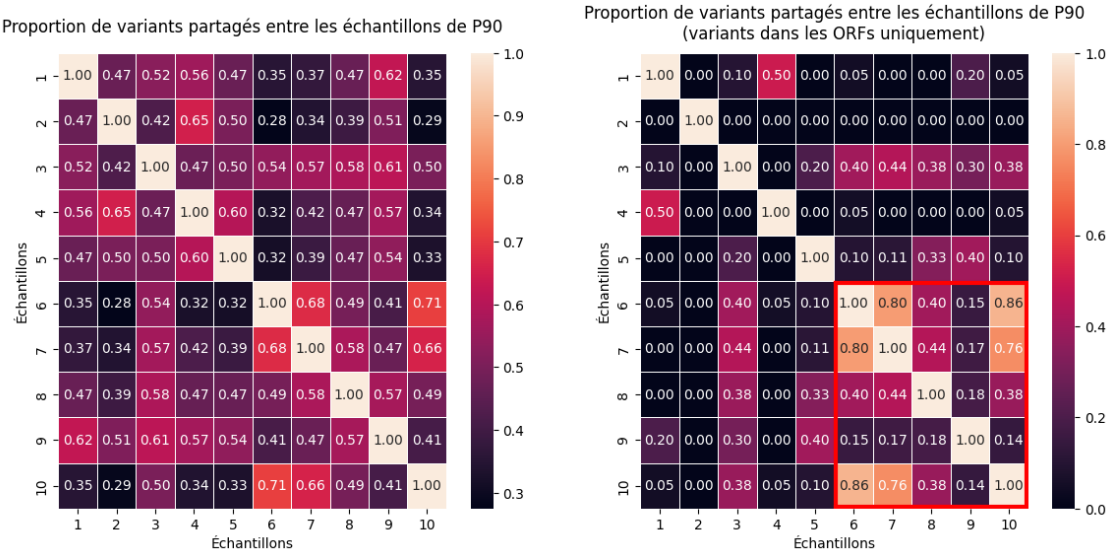
Dans la seconde matrice on se concentre sur les variations présentent uniquement dans les ORFs. On remarque un cluster entre les échantillons 6 à 10, comfirmant que les délétions trouvés dans ces échantillons sont récurrentes et renforcant l'hypothèse qu'elles ont un lien avec le choc thermique. Comme expliqué précédemment, il faut faire attention à l'interprétation des similarités entre les échantillons 1 à 5 (froid) puisque ceux ci présentent très peu de mutations dans des ORFs, les valeurs pourraientt alors être biaisé et traduire une absence de mutations dans les ORFs plutot qu'une hétérogénéité du groupe.

Enfin on remarque les échantillons 3 et 9 qui se démarquent nettement de leur groupe. Hypothèses: - un autre mécanisme entre en jeu (par exemple un autre variants, plus rare à permis de contrer le potentiel effet délétaire des délétions) - le séquencage à pu révéler des variants dans l'échantillon 3 mais pas dans les 3 autres, ce qui voudrais dire que ces variants ne serait pas nécéssairement corrélés au choc thermique, plutot un problème expérimental (échantillonage ? séquencage ?) -> regarder en détail l'échantillon 3 pour voir si il se démarque des autres (profondeur de l'alignement, distribution des variants, proportions de tags assignés, ect...) - certains adn provenant de P90-3 ont étés échangés (ou leurs tags) avec des adn provenant de P90-9 -> qui a préparé P90-3 lol ?

[46]:
```
def pairwise_similarity(sv, samples, key, thresold=0):
    grouped = sv.groupby(["group"], observed=True)
    sims = np.ones(shape=(len(samples), len(samples)), dtype=np.float64)
    for i, s1 in enumerate(samples):
```

```
        for j in range(0, i):
            s2 = samples[j]
            from_i_or_j = [g for g in grouped.indices if s1 in grouped.
 ↪get_group(g)[key].values or s2 in grouped.get_group(g)[key].values]
            from_i_and_j = [g for g in grouped.indices if s1 in grouped.
 ↪get_group(g)[key].values and s2 in grouped.get_group(g)[key].values]
            sims[i][j] = len(from_i_and_j) / len(from_i_or_j) if␣
 ↪len(from_i_or_j) > 0 else 0
            sims[i][j] = sims[i][j] if sims[i][j] >= thresold else 0
            sims[j][i] = sims[i][j]
    return sims
```

[47]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

samples = data["sample"].astype(int).unique()
samples.sort()

sim = pairwise_similarity(data_oi, samples, "sample")
sns.heatmap(sim, annot=True, linewidth=.4, fmt=".2f", xticklabels=samples,␣
 ↪yticklabels=samples, ax=ax1)
ax1.set_xlabel("Échantillons")
ax1.set_ylabel("Échantillons")
ax1.set_title(f"Proportion de variants partagés entre les échantillons de␣
 ↪P{ioi}", pad=15)

sim = pairwise_similarity(data_oi_orf, samples, "sample")
sns.heatmap(sim, annot=True, linewidth=.4, fmt=".2f", xticklabels=samples,␣
 ↪yticklabels=samples, ax=ax2)
ax2.set_xlabel("Échantillons")
ax2.set_ylabel("Échantillons")
ax2.set_title(f"Proportion de variants partagés entre les échantillons de␣
 ↪P{ioi}\n(variants dans les ORFs uniquement)", pad=10)

ax2.plot([5.05, 5.05], [5.05, 9.97], 'r-', lw=3)
ax2.plot([9.95, 9.95], [5.05, 9.97], 'r-', lw=3)
ax2.plot([5.05, 9.95], [9.97, 9.97], 'r-', lw=3)
ax2.plot([5.05, 9.95], [5.05, 5.05], 'r-', lw=3)

plt.show()
```



4.4 Recherche des variants discriminants

Si des variants sont corrélés avec le choc thermique, ils ont probablement un effet significatif dans un des deux groupes. La solution la plus direct est de rechercher les variants qui sont dans des ORFs et qui ne sont présent que dans le groupe chaud ou le groupe froid.

Cette méthode présente plusieurs inconvénients. Elle est trop restrictive et peu rater des variants présent en fréquence significativement différentes. Par exemple si un variants et présent 1 fois dans le chaud et 5 fois dans le froid il ne sera pas détecté malgrès une différence significative, alors que si il est présent 0 fois dans le chaud et 1 fois dans le froid, il sera détecté (situation beaucoup moins significative).

Il faudrais alors mettre en place un test statistique plus robuste qui nous permette de dire si l'apparition d'une variation est corrélé avec le choc thermique (fisher ? khi2 ? …).

Suite à cela nous avons choisis d'investiguer le variant de l'ORF78 qui semble être la plus importante.

[48]:
```
uniques = data_oi_orf.groupby(["group"], observed=True).filter(lambda x: not␣
  ↪("heat" in x["choc"].values and "cold" in x["choc"].values))
uniques[["id", "pos", "af", "orfs"]].head(17)
```

[48]:

index	id	pos	af	\
54	Sniffles2.DEL.1D5S0	9457	0.594	
733	Sniffles2.DEL.2D2S0	47346	0.490	
877	Sniffles2.DEL.29CS0	60281	1.000	
880	Sniffles2.DEL.2E3S0	60281	1.000	
883	Sniffles2.DEL.329S0	60281	1.000	
887	Sniffles2.DEL.274S0	60281	1.000	
1277	Sniffles2.DEL.2F2S0	127195	0.070	

```
1284   Sniffles2.DEL.32ES0  127195  0.054
1318   Sniffles2.DEL.302S0  133271  0.857
1320   Sniffles2.DEL.332S0  133271  0.783
1325   Sniffles2.DEL.33DS0  133271  0.840
1345   Sniffles2.DEL.318S0  143509  0.905
1347   Sniffles2.DEL.349S0  143509  0.852
1352   Sniffles2.DEL.35FS0  143509  0.891
1403   Sniffles2.DEL.323S0  152722  0.073
1404   Sniffles2.DEL.357S0  152722  0.067
1406   Sniffles2.DEL.379S0  152722  0.050
```

index	orfs
54	[CyHV3_ORF5_1]
733	[CyHV3_ORF25, CyHV3_ORF26, CyHV3_ORF27]
877	[CyHV3_ORF40]
880	[CyHV3_ORF40]
883	[CyHV3_ORF40]
887	[CyHV3_ORF40]
1277	[CyHV3_ORF68]
1284	[CyHV3_ORF68]
1318	[CyHV3_ORF69]
1320	[CyHV3_ORF69]
1325	[CyHV3_ORF69]
1345	[CyHV3_ORF78]
1347	[CyHV3_ORF78]
1352	[CyHV3_ORF78]
1403	[CyHV3_ORF82]
1404	[CyHV3_ORF82]
1406	[CyHV3_ORF82]

5 Analyse de l'échantillon 10

Remarque : Cette analyse peut être lancé sur n'importe quel échantillon en modifiant la variable **soi** ci dessous.

On veux comparer les différent passages d'un même échantillon pour voir si il y a un évolution particulière dans la distribution des variants. C'est quasi identique à l'anlayse précédente, on compare les passage d'un échantillon donné au lieu de comparer les échantillons d'un passage donc je ne vais pas trop rentrer dans les détails

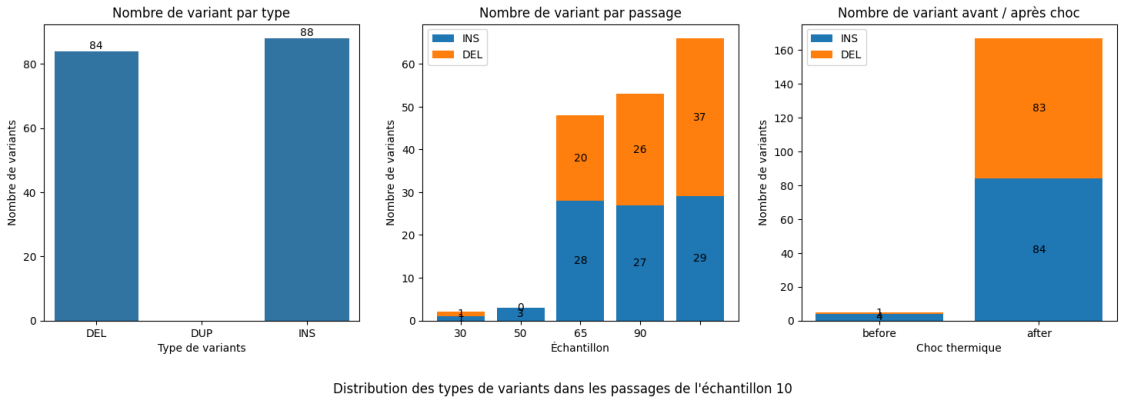
[49]:
```
soi = 10
data_soi = data[data["sample"] == soi]
data_soi_orf = data_soi[data_soi["orfs"].astype(bool)]
```

5.1 Distribution des variants

On compte le nombre de variant par type dans tous les passages de l'échantillon 10 (P15-10 à P90-10). Sur le 2ème plot on compte par passage, et sur le 3ème plot on distingue avant (< P30) et après (>= P30) le choc.

[50]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.countplot(data=data_soi, x="svtype", ax=ax1)
ax1.bar_label(container = ax1.containers[0], fontsize=10)
ax1.set_ylabel('Nombre de variants')
ax1.set_xlabel('Type de variants')
ax1.set_title(f"Nombre de variant par type")

type_count_sample = data_soi.groupby(["svtype", "iteration"], observed=True).
 ↪size()
types = data_soi["svtype"].unique()
iters = data["iteration"].astype(int).unique()
iters.sort()
bottom = np.zeros(len(iters))
for t in types:
    gc = [type_count_sample[t][i] if i in type_count_sample[t] else 0 for i in␣
 ↪iters]
    p = ax2.bar(range(len(iters)), gc, label=t, bottom=bottom)
    bottom += gc
    ax2.bar_label(p, label_type='center')

ax2.set_ylabel('Nombre de variants')
ax2.set_xlabel('Échantillon')
ax2.set_xticklabels([0] + iters)
ax2.set_title(f"Nombre de variant par passage")
ax2.legend()


data_soi["period"] = data_soi.apply(lambda x: "before" if x["iteration"] <= 30␣
 ↪else "after", axis=1)
type_count_time = data_soi.groupby(["svtype", "period"], observed=True).size()
types = data_soi["svtype"].unique()
periods = ["before", "after"]
bottom = np.zeros(len(periods))
for t in types:
    gc = [type_count_time[t][p] if p in type_count_time[t] else 0 for p in␣
 ↪periods]
    p = ax3.bar(periods, gc, label=t, bottom=bottom)
    bottom += gc
    ax3.bar_label(p, label_type='center')

ax3.set_ylabel('Nombre de variants')
```

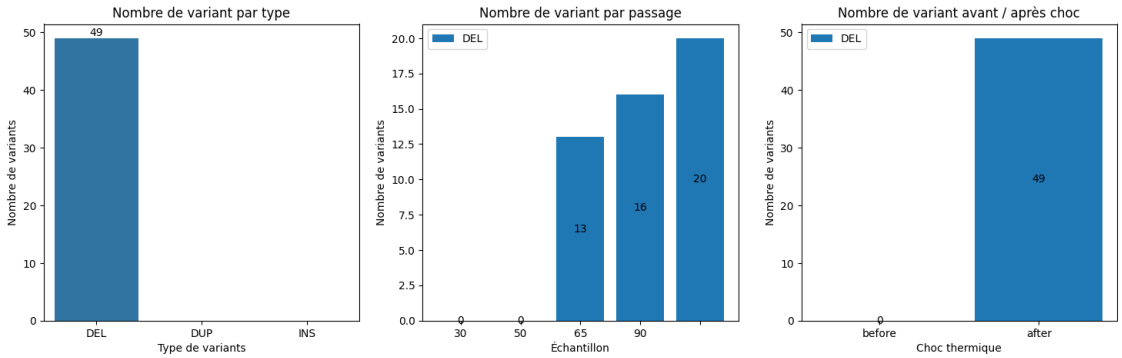
```
ax3.set_xlabel('Choc thermique')
ax3.set_title(f"Nombre de variant avant / après choc")
ax3.legend()

plt.suptitle(f"Distribution des types de variants dans les passages de␣
 ↪l'échantillon {soi}", y=-0.05)
plt.show()
```



On réitère avec les variant qui chevauchent des ORFs

[51]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.countplot(data=data_soi_orf, x="svtype", ax=ax1)
ax1.bar_label(container = ax1.containers[0], fontsize=10)
ax1.set_ylabel('Nombre de variants')
ax1.set_xlabel('Type de variants')
ax1.set_title(f"Nombre de variant par type")

type_count_sample = data_soi_orf.groupby(["svtype", "iteration"],␣
 ↪observed=True).size()
types = data_soi_orf["svtype"].unique()
iters = data["iteration"].astype(int).unique()
iters.sort()
bottom = np.zeros(len(iters))
for t in types:
    gc = [type_count_sample[t][i] if i in type_count_sample[t] else 0 for i in␣
 ↪iters]
    p = ax2.bar(range(len(iters)), gc, label=t, bottom=bottom)
    bottom += gc
    ax2.bar_label(p, label_type='center')

ax2.set_ylabel('Nombre de variants')
ax2.set_xlabel('Échantillon')
```

```
ax2.set_xticklabels([0] + iters)
ax2.set_title(f"Nombre de variant par passage")
ax2.legend()


data_soi_orf["period"] = data_soi_orf.apply(lambda x: "before" if␣
 ↪x["iteration"] <= 30 else "after", axis=1)
type_count_time = data_soi_orf.groupby(["svtype", "period"], observed=True).
 ↪size()
types = data_soi_orf["svtype"].unique()
periods = ["before", "after"]
bottom = np.zeros(len(periods))
for t in types:
    gc = [type_count_time[t][p] if p in type_count_time[t] else 0 for p in␣
 ↪periods]
    p = ax3.bar(periods, gc, label=t, bottom=bottom)
    bottom += gc
    ax3.bar_label(p, label_type='center')

ax3.set_ylabel('Nombre de variants')
ax3.set_xlabel('Choc thermique')
ax3.set_title(f"Nombre de variant avant / après choc")
ax3.legend()

plt.suptitle(f"Distribution des types de variants dans les passages de␣
 ↪l'échantillon {soi} (variants dans les ORFs uniquement)", y=-0.05)
plt.show()
```

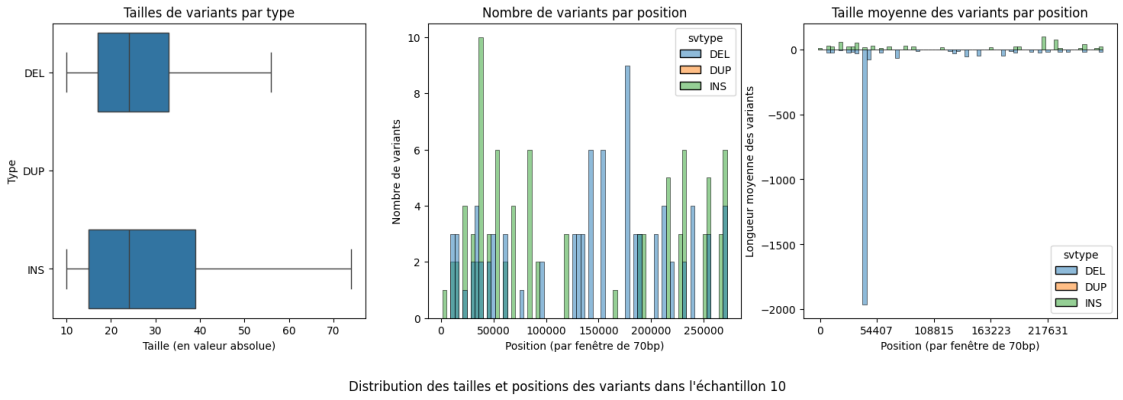


Distribution des types de variants dans les passages de l'échantillon 10 (variants dans les ORFs uniquement)

5.2 Tailles et positions des variants

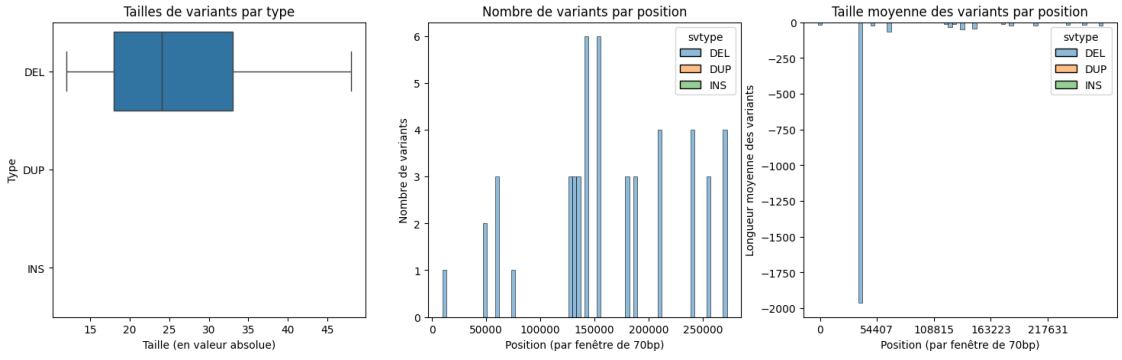
Dans l'ensemble des passages de l'échantillon 10, on mesure la taille des variants et leur répartition le long du génome.

[52]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.boxplot(data=data_soi, x=abs(data_soi["svlen"]), y="svtype",␣
 ↪showfliers=False, ax=ax1)
ax1.set_xlabel("Taille (en valeur absolue)")
ax1.set_ylabel("Type")
ax1.set_title("Tailles de variants par type")

nbins = 70
sns.histplot(data=data_soi, x="pos", hue="svtype", bins=nbins, ax=ax2)
ax2.set_xlabel(f"Position (par fenêtre de {nbins}bp)")
ax2.set_ylabel("Nombre de variants")
ax2.set_title("Nombre de variants par position")

max_pos = max(data_soi["pos"])
data_soi.loc[:, 'pos_bin'] = pd.cut(data_soi['pos'], nbins, labels=range(nbins))
data_mean = data_soi.groupby(['pos_bin', 'svtype'], as_index=False,␣
 ↪observed=True)['svlen'].mean()
sns.histplot(data=data_mean, x="pos_bin", hue="svtype", weights='svlen',␣
 ↪bins=nbins, ax=ax3)
ax3.set_xlabel(f"Position (par fenêtre de {nbins}bp)")
ax3.set_ylabel("Longueur moyenne des variants")
ax3.set_xticks([i * (nbins / 5) for i in range(5)], labels=[int(i * (max_pos /␣
 ↪5)) for i in range(5)])
ax3.set_title("Taille moyenne des variants par position")

plt.suptitle(f"Distribution des tailles et positions des variants dans␣
 ↪l'échantillon {soi}", y=-0.05)
plt.show()
```



On réitère en filtrant les variants dans des ORFs

[53]:
```
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 5))

sns.boxplot(data=data_soi_orf, x=abs(data_soi_orf["svlen"]), y="svtype",␣
 ↪showfliers=False, ax=ax1)
ax1.set_xlabel("Taille (en valeur absolue)")
ax1.set_ylabel("Type")
ax1.set_title("Tailles de variants par type")

nbins = 70
sns.histplot(data=data_soi_orf, x="pos", hue="svtype", bins=nbins, ax=ax2)
ax2.set_xlabel(f"Position (par fenêtre de {nbins}bp)")
ax2.set_ylabel("Nombre de variants")
ax2.set_title("Nombre de variants par position")

max_pos = max(data_soi_orf["pos"])
data_soi_orf.loc[:, 'pos_bin'] = pd.cut(data_soi_orf['pos'], nbins,␣
 ↪labels=range(nbins))
data_mean = data_soi_orf.groupby(['pos_bin', 'svtype'], as_index=False,␣
 ↪observed=True)['svlen'].mean()
sns.histplot(data=data_mean, x="pos_bin", hue="svtype", weights='svlen',␣
 ↪bins=nbins, ax=ax3)
ax3.set_xlabel(f"Position (par fenêtre de {nbins}bp)")
ax3.set_ylabel("Longueur moyenne des variants")
ax3.set_xticks([i * (nbins / 5) for i in range(5)], labels=[int(i * (max_pos /␣
 ↪5)) for i in range(5)])
ax3.set_title("Taille moyenne des variants par position")

plt.suptitle(f"Distribution des tailles et positions des variants dans␣
 ↪l'échantillon {soi} (variants dans les ORFs uniquement)", y=-0.05)
plt.show()
```



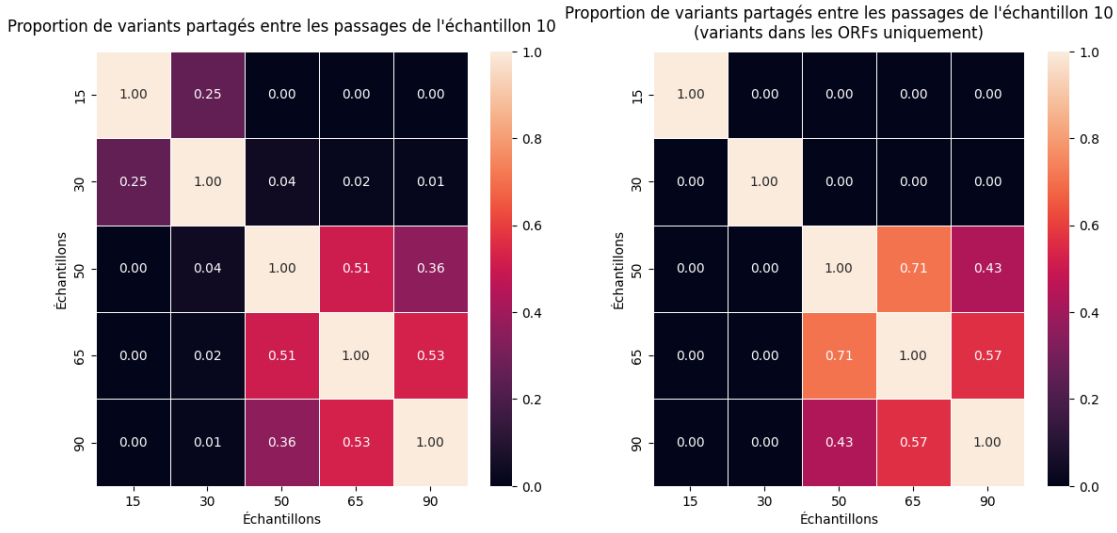
Distribution des tailles et positions des variants dans l'échantillon 10 (variants dans les ORFs uniquement)

5.3 Similarité entre les passages

Mesure de similarité par paire de passage.

On remarque bien ici une différence entre avant le choc et après le choc (cependant on voit précedemment qu'il y a très peu de variants avants le choc donc attention aux potentiel biais)

[54]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

iters = data["iteration"].astype(int).unique()
iters.sort()

sim = pairwise_similarity(data_soi, iters, "iteration")
sns.heatmap(sim, annot=True, linewidth=.4, fmt=".2f", xticklabels=iters,␣
 ↪yticklabels=iters, ax=ax1)
ax1.set_xlabel("Échantillons")
ax1.set_ylabel("Échantillons")
ax1.set_title(f"Proportion de variants partagés entre les passages de␣
 ↪l'échantillon {soi}", pad=15)

sim = pairwise_similarity(data_soi_orf, iters, "iteration")
sns.heatmap(sim, annot=True, linewidth=.4, fmt=".2f", xticklabels=iters,␣
 ↪yticklabels=iters, ax=ax2)
ax2.set_xlabel("Échantillons")
ax2.set_ylabel("Échantillons")
ax2.set_title(f"Proportion de variants partagés entre les passages de␣
 ↪l'échantillon {soi}\n(variants dans les ORFs uniquement)", pad=10)

ax2.plot([5.05, 5.05], [5.05, 9.97], 'r-', lw=3)
ax2.plot([9.95, 9.95], [5.05, 9.97], 'r-', lw=3)
ax2.plot([5.05, 9.95], [9.97, 9.97], 'r-', lw=3)
ax2.plot([5.05, 9.95], [5.05, 5.05], 'r-', lw=3)

plt.show()
```



6 Perspectives

- test d'homogénéité formel pour comparer deux ensembles de variants
- test de corrélation formel pour déterminer si:
  - l'apparition d'un variant est corrélé avec le choc thermique -> on peut soit comparer les échantillons 1-5 vs 6-10 dans chaque passage, soit comparer le passage 30 vs les autres pour chaque échantillons, ...
  - la corrélation entre la présence absence de 2 variants pour déterminer si ils ont des effets complémentaires
- il s'agit de pouvoir réaliser l'analyse qui à ici été faite graphiquement, de facon statistique avec des métrique plus facilement interprétables
- investiguer échantillons 3 et 9
- considérer un seuil de similarité entre échantillon plus faible pour les considérer identiques (100% pour le moment)
- le choc thermique peut il être à la source de l'apparition / la disparition d'un variant ? plus dans le chaud ou dans le froid ?