Handheld VESAD Analysis

June 20, 2018

The analysis of the experiment data is also accessible online: https://github.com/NormandErwan/HandheldVesadAnalysis/blob/master/Handheld%20VESAD%20Analysis.ipynb.

0.1 1. Data preparation

Configuration :

```
In [1]: # Imports
        import numpy as np
        import pandas as pd
        from pandas.api.types import CategoricalDtype
        import itertools

        import seaborn as sns
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import matplotlib.ticker as ticker
        import matplotlib.patches as patches

        from scipy import stats
        import statsmodels.api as sm
        from statsmodels.formula.api import ols
        from statsmodels.stats.multicomp import (MultiComparison, pairwise_tukeyhsd)
        from statsmodels.stats.multitest import multipletests
        from statsmodels.stats.libqsturng import psturng

        from ast import literal_eval
        from os import listdir
        from os.path import join

        from IPython.html.services.config import ConfigManager
```

```
C:\Users\Erwan\Miniconda\envs\master-thesis\lib\site-packages\IPython\html.py:14: ShimWarning: The
  "`IPython.html.widgets` has moved to `ipywidgets`.", ShimWarning)
```

```
In [2]: # Notebook configuration
        %matplotlib inline

        # Ruler
```

```python
ip = get_ipython()
cm = ConfigManager(parent=ip)
cm.update('notebook', {"ruler_column": [80]})

# Style and size of the figures
subplotsize = (5, 4)
legend_fontsize = sns.plotting_context('notebook')['axes.labelsize']
legend_title_fontsize = legend_fontsize + 1
sns.set(context='notebook', style='whitegrid', font_scale=1.25,
        rc={'legend.fontsize': legend_fontsize, 'savefig.dpi': 300,
            'savefig.bbox': 'tight'})

# Render the plots with this language
#language = 'English'
language = 'Français'
```

Data are loaded in the following variables: - Participants information, from the questionary before the trials: `participants` - The summary of the trials of the participants: `raw_trials` - Trials averaged so that each participants ends up with a single observation (there was two trials per condition per participant) (Tip 9 of Dragicevic (2016)) : `trials` - The detailed measures of the trials: `trial_details` - The ranks of the participants from the post-questionary: `ranks`

```python
In [3]: participants = pd.read_csv('participants.csv')
        raw_trials = pd.read_csv('participant_trials.csv')
        trials_for_anova = pd.read_csv('participant_trials.csv')
        ranks = pd.read_csv('participant_ranks.csv')
```

```python
In [4]: # Detailled trials are long to load, keep commented if unused

        #trial_details = []
        #for file in listdir('.'):
        #    if file.endswith('details.csv'): # The trial files end with *-details.csv
        #        details.append(pd.read_csv(file))
        #trial_details = pd.concat(details, ignore_index=True)
```

Creates independent variables lists (`participants_ivs`, `trials_ivs`) and dependent variables lists (`ranks_dvs`, `trials_dvs`) to make easier to use algorithms and to plot figures.

```python
In [5]: # Participants IVs
        if language == 'Français':
            participants_iv_labels = ['Numéro participant', 'Sexe',
                                      'Porte des lunettes', 'Porte des lentilles',
                                      'Est daltonien', 'Intervalle d\'âge',
                                      'Main dominante', 'Main utilisée pour la souris',
                                      'Activité principale',
                                      'Utilisation de l\'ordinateur (heures/jours)',
                                      'Logiciels 3D utilisés',
                                      'Visiocasques RV/RA utilisés',
                                      'Techniques d\'interactions RV/RA utilisées']
        else:
            participants_iv_labels = ['Participant Id', 'Sex', 'Has Glasses',
```

```
                                'Has Contact Lenses', 'Is Color Blind',
                                'Age Class', 'Dominant Hand',
                                'Hand Used for Mouse', 'Activity',
                                'Computer Hours per Day', '3D Softwares Used',
                                'HMD Used',
                                'Known Interactions Techniques on HMD']

participants_ivs = pd.Series(data=participants_iv_labels,
                             index=participants.columns)
```

In [6]:
```
# Trials IVs
if language == 'Français':
    trials_iv_labels = ['Technique', 'Taille du texte', 'Distance', 'Groupe',
                        'Méthode d\'entrée', 'Méthode d\'affichage']
else:
    trials_iv_labels = ['Technique', 'Text Size', 'Distance', 'Ordering',
                        'Technique Input', 'Technique Output']

trials_ivs = ['technique', 'text_size',
              'distance', 'ordering', 'input', 'output']
trials_ivs = pd.DataFrame(columns=trials_ivs,\
                          index=['label', 'categorical', 'palette'])

default_palette = sns.color_palette('Set2', 8)

for iv_index, iv_label in zip(trials_ivs.columns, trials_iv_labels):
    iv_categories = raw_trials.sort_values([iv_index + '_id'])\
    .drop_duplicates(iv_index)[iv_index]
    iv_categorical = pd.Categorical(iv_categories, iv_categories, ordered=True)
    trials_ivs[iv_index] = [iv_label, iv_categorical, default_palette]

technique_palette = [default_palette[1], default_palette[0], default_palette[2]]
trials_ivs.at['palette', 'technique'] = technique_palette
trials_ivs.at['palette', 'text_size'] = sns.light_palette(
    default_palette[6], 3)[1:3]  # Brown paired colors
trials_ivs.at['palette', 'distance'] = sns.light_palette(
    default_palette[4], 3)[1:3]  # Green paired colors
trials_ivs.at['palette', 'ordering'] = technique_palette
trials_ivs.at['palette', 'input'] = [default_palette[5], default_palette[2]]
trials_ivs.at['palette', 'output'] = [default_palette[0],
                                      sns.color_palette('muted')[5]]
```

In [7]:
```
# Trials DVs
if language == 'Français':
    trials_dv_labels = ['Temps de complétion (s)', 'Sélections',
                        'Temps en sélection', 'Distance 3D en sélection',
                        'Distance en sélection', 'Déselections', 'Erreurs',
                        'Disques classés', 'Défilements', 'Temps de défilement',
                        'Distance 3D de défilement', 'Distance de défilement',
                        'Zooms', 'Temps de zoom', 'Distance 3D de zoom',
                        'Distance de zoom', 'Mouvements tête-téléphone',
```

```
                               'Distance relative tête-téléphone']
        else:
            trials_dv_labels = ['Task Completion Time (s)', 'Selections',
                                'Selection Time', 'Selection Distance',
                                'Selection Distance on Grid', 'Deselections', 'Errors',
                                'Items Classified', 'Pans', 'Pan Time', 'Pan Distance',
                                'Pan Distance on Grid', 'Zooms', 'Zoom Time',
                                'Zoom Distance', 'Zoom Distance on Grid',
                                'Phone-Head Motion', 'Signed Phone-Head Motion']

        trials_dvs = raw_trials.loc[:, 'total_time':'signed_head_phone_distance'].columns
        trials_dvs = pd.Series(data=trials_dv_labels, index=trials_dvs)
```

```
In [8]: # Ranks DVs
        if language == 'Français':
            ranks_dv_labels = ['Facile à comprendre', 'Mentalement facile à utiliser',
                               'Physiquement facile à utiliser', 'Rapidité', 'Réussite',
                               'Frustration', 'Préférence']
        else:
            ranks_dv_labels = ['Easy to Understand', 'Mentally Easy to Use',
                               'Physically Easy to Use', 'Subjective Speed',
                               'Subjective Performance', 'Frustration', 'Preference']

        ranks_dv_scales = [pd.Categorical(list(range(1, 6)), list(
            range(1, 6)), ordered=True)] * len(ranks_dv_labels)

        RdYlBu = sns.color_palette('RdYlBu', 5)
        ranks_dv_palettes = [sns.color_palette('RdYlBu', 5)] * len(ranks_dv_labels)

        ranks_dvs = ranks.loc[:, 'easy_understand':'preference'].columns
        ranks_dvs = pd.DataFrame(data=[ranks_dv_labels, ranks_dv_scales,
                                       ranks_dv_palettes],
                                 columns=ranks_dvs, index=['label', 'scale', 'palette'])

        ranks_dvs.at['scale', 'preference'] = pd.Categorical(
            list(range(1, 4)), list(range(1, 4)), ordered=True)
        ranks_dvs.at['palette', 'preference'] = [RdYlBu[4], RdYlBu[2], RdYlBu[0]]
```

```
In [9]: # Shortcuts
        participant_id = participants_ivs['participant_id']
        technique = trials_ivs['technique']
        text_size = trials_ivs['text_size']
        distance = trials_ivs['distance']
        ordering = trials_ivs['ordering']
        iv_input = trials_ivs['input']
        iv_output = trials_ivs['output']
```

Clean the data:

```
In [10]: # Set better and translated columns to participants, trials and ranks
         participants.columns = participants_ivs
```

```python
columns = []
for column in raw_trials.columns:
    if (column in participants_ivs.index):
        columns.append(participants_ivs[column])
    elif (column in trials_ivs.columns):
        columns.append(trials_ivs[column]['label'])
    elif (column in trials_dvs.index):
        columns.append(trials_dvs[column])
    else:
        columns.append(column)

raw_trials.columns = columns

ranks.columns = [participant_id, ordering['label'],
                 technique['label']] + ranks_dvs.loc['label', :].tolist()
```

In [11]:
```python
# Set the participant_id column as the index in participants
participants.set_index(participants_ivs['participant_id'], inplace=True)
```

In [12]:
```python
# Some participants are non valid or don't have complete measures
non_valid_participants = [0, 4]
participants = participants[~participants.index.isin(non_valid_participants)]
ranks = ranks[~ranks[participants_ivs['participant_id']]
              .isin(non_valid_participants)]

incomplete_trials_participant_ids = [0, 4]
raw_trials = raw_trials[~raw_trials[participants_ivs['participant_id']]
                        .isin(incomplete_trials_participant_ids)
                        ].reset_index(drop=True)
trials_for_anova = trials_for_anova[~trials_for_anova['participant_id']
                                    .isin(incomplete_trials_participant_ids)
                                    ].reset_index(drop=True)
```

In [13]:
```python
# Some participants have wrong head phone mesures
for head_distance_column in ['absolute_head_phone_distance',
                             'signed_head_phone_distance']:
    raw_trials.loc[raw_trials[trials_dvs[head_distance_column]] == 0,
                   trials_dvs['absolute_head_phone_distance']] = np.nan
    trials_for_anova.loc[trials_for_anova[head_distance_column] == 0,
                         head_distance_column] = np.nan
```

In [14]:
```python
# Setup categorical columns participants, trials and ranks
participants[ordering['label']] = ranks.groupby(
    participant_id)[ordering['label']].first()
participants[ordering['label']] = participants[ordering['label']].astype(
    ordering['categorical'])

for iv_index in trials_ivs.columns:
    iv = trials_ivs[iv_index]
    raw_trials[iv['label']] = raw_trials[iv['label']].astype(iv['categorical'])
```

```
            trials_for_anova[iv_index] = raw_trials[iv['label']]

        ranks[technique['label']] = ranks[technique['label']].astype(
            technique['categorical'])
        ranks[ordering['label']] = ranks[ordering['label']].astype(
            ordering['categorical'])

        # Rename categories
        if language == 'Français':
            technique['categorical'].categories = ['Téléphone', 'VESAD tactile', 'VESAD']
            text_size['categorical'].categories = ['Grand', 'Petit']
            distance['categorical'].categories = ['Proche', 'Loin']
            ordering['categorical'].categories = ['Groupe 1', 'Groupe 2', 'Groupe 3']
            iv_input['categorical'].categories = ['Tactile', 'Autour du téléphone']
            iv_output['categorical'].categories = ['Téléphone seul', 'Téléphone étendu']
        else:
            ordering['categorical'].categories = ['Group 1', 'Group 2', 'Group 3']

        # Set renamed categories to data
        participants[trials_ivs['ordering']['label']
                     ].cat.categories = ordering['categorical'].categories

        for iv_index in trials_ivs.columns:
            iv = trials_ivs[iv_index]
            raw_trials[iv['label']].cat.categories = iv['categorical'].categories

        ranks[technique['label']].cat.categories = technique['categorical'].categories
        ranks[ordering['label']].cat.categories = ordering['categorical'].categories
```

```
In [15]: # Eval the arrays in some dvs
         def eval_if_str(data):
             return literal_eval(data) if isinstance(data, str) else data

         raw_trials['grid_config'] = raw_trials['grid_config'].apply(eval_if_str)
```

```
In [16]: # Create trials
         trials_groupby_ivs = [participants_ivs['participant_id']]\
                              + trials_ivs.loc['label', :].tolist()
         trials = raw_trials.groupby(trials_groupby_ivs, observed=True)\
                  .mean().reset_index()
```

Utilities:

```
In [17]: # Figure labels in the selected language
         labels = pd.Series()

         if language == 'Français':
             labels['category'] = 'Catégorie'
             labels['count'] = 'Nombre total'
             labels['distance'] = 'Distance moyenne (m)'
             labels['dv'] = 'Variable dépendante'
```

```python
            labels['iv'] = 'Variable indépendante'
            labels['iv_value'] = 'Valeur VI'
            labels['mean_difference'] = 'Différence des moyennes'
            labels['mean_difference_percentage'] = 'Différence des moyennes (%)'
            labels['mean_rank'] = 'Note moyenne'
            labels['preferences'] = ['Premier', 'Deuxième', 'Troisième']
            labels['p_value'] = 'Valeur p'
            labels['question'] = 'Question'
            labels['rank'] = 'Note'
            labels['time'] = 'Temps moyen (s)'
            labels['t_statistic'] = 'Statistique T'
            labels['votes'] = 'Participants'
        else:
            labels['category'] = 'Category'
            labels['count'] = 'Count'
            labels['distance'] = 'Mean Distance (m)'
            labels['dv'] = 'Dependent Variable'
            labels['iv'] = 'Independent Variable'
            labels['iv_value'] = 'IV Value'
            labels['mean_difference'] = 'Mean Difference'
            labels['mean_difference_percentage'] = 'Mean Difference Percentage'
            labels['mean_rank'] = 'Mean Rank'
            labels['preferences'] = ['First', 'Second', 'Third']
            labels['p_value'] = 'p-value'
            labels['question'] = 'Question'
            labels['rank'] = 'Rank'
            labels['time'] = 'Mean Time (s)'
            labels['t_statistic'] = 'T statistic'
            labels['votes'] = 'Participants'
```

In [18]:
```python
def mean_ci(x, which=95, n_boot=1000):
    """Returns the confidence interval of the mean"""
    x2 = [i for i in x if not np.isnan(i)]
    boots = sns.algorithms.bootstrap(x2, n_boot=n_boot)
    return sns.utils.ci(boots, which=which)
```

In [19]:
```python
def print_mean_ci(x, ci_which=95):
    """Returns a string containing the mean with the CI of x"""
    ci1, ci2 = mean_ci(x, which=ci_which)
    return '{:.2f} [{:.2f}, {:.2f}]'.format(np.mean(x), ci1, ci2)
```

In [20]:
```python
def mean_difference(a, b):
    """Returns the mean difference value and percentage between a and b"""
    mean_diff = np.mean(a) - np.mean(b)
    mean_diff_percentage = mean_diff / np.mean(b) * 100
    return (mean_diff, mean_diff_percentage)
```

In [21]:
```python
def p_values_correction(data, alpha=0.05, correction_method='fdr_bh'):
    if correction_method != None:
        reject, p_values_corrected, a1, a2 =\
            multipletests(data[labels['p_value']].tolist(), alpha=alpha,
```

```
                                method=correction_method)
        data[labels['p_value']] = p_values_corrected
```

```
In [22]: def subplots(nsubplots, ncols_max=2, subplotsize=subplotsize, *plt_args):
             ncols = min(ncols_max, nsubplots)
             nrows = ((nsubplots - 1) // ncols) + 1
             fig, axs = plt.subplots(nrows=nrows, ncols=ncols,\
                                     figsize=(subplotsize[0]*ncols, subplotsize[1]*nrows),
                                     *plt_args)

             if nrows == 1 and ncols == 1:
                 axs = [axs]
             elif nrows >= 2 and ncols >= 2:
                 axs = [ax for ax_row in axs for ax in ax_row]

             for ax in axs[::-1][0:len(axs) - nsubplots]:
                 fig.delaxes(ax)

             return (fig, axs)
```

```
In [23]: def fix_legend_fontsize(ax_legend, fontsize=legend_title_fontsize):
             plt.setp(ax_legend.get_title(), fontsize=fontsize)
```

```
In [24]: def config_legend(ax, iv_id, fontsize=legend_title_fontsize):
             legend = ax.legend(title=trials_ivs.at['label', iv_id], frameon=True,
                                loc='center left', bbox_to_anchor=(1, 0.5))
             fix_legend_fontsize(legend)
             return legend
```

```
In [25]: def log_mean(x):
             return np.exp(np.mean(np.log(x)))
```

0.2 2. Participant ranks

Some functions for the analysis:

```
In [26]: def get_ranks_count(iv_index, dv_index):
             iv, dv = trials_ivs[iv_index], ranks_dvs[dv_index]

             ranks_counts_index = pd.MultiIndex.from_product([iv['categorical'],
                                                               dv['scale']],
                                                              names=[iv['label'],
                                                                     labels['rank']])
             # Zero counts by default and gets the counts
             default_counts = pd.Series(0, index=ranks_counts_index)
             ranks_counts = ranks.groupby([iv['label'], dv['label']]).size()

             # Merge and remove duplicated defaults
             ranks_counts = pd.concat([ranks_counts, default_counts])
             ranks_counts = ranks_counts[~ranks_counts.index.duplicated(keep='first')]

             ranks_counts.sort_index(inplace=True)
```

```
        ranks_counts.index = ranks_counts_index # Restore the index
        return ranks_counts
```

```
In [27]: def cumulated_barplot(data, palette, **args):
             for row_id, row in data.iloc[::-1].iterrows():
                 sns.barplot(y=data.columns, x=row, label=row_id,
                             color=palette[row_id-1], orient='h', **args)
```

```
In [28]: def plot_ranks_distributions(iv_id, dv_ids):
             iv = trials_ivs[iv_id]

             fig, axs = subplots(len(dv_ids))
             for dv_id, ax in zip(dv_ids, axs):
                 dv = ranks_dvs[dv_id]

                 cumulated_ranks_count = get_ranks_count(iv_id, dv_id).unstack(level=0)\
                                         .cumsum()
                 cumulated_barplot(cumulated_ranks_count, palette=dv['palette'], ax=ax)
                 ax.set(xlabel=labels['votes'], xlim=(0, cumulated_ranks_count.max()[0]))
                 ax.xaxis.set_major_locator(ticker.MultipleLocator(2)) # Fix the axis ticks

                 ax_handles, ax_labels = ax.get_legend_handles_labels()
                 legend = ax.legend(ax_handles[::-1], ax_labels[::-1], frameon=True,
                                    loc='lower center', bbox_to_anchor=(0.5, 1),
                                    mode=None, ncol=len(dv['scale']), title=dv['label'],
                                    fontsize=legend_fontsize-2)
                 fix_legend_fontsize(legend)

             fig.tight_layout(h_pad=1) # Add padding to avoir legend and labels overlap
             return (fig, axs)
```

```
In [29]: def plot_ranks(iv_id, dv_ids, estimator=np.mean):
             iv = trials_ivs[iv_id]

             fig, axs = subplots(len(dv_ids))
             for dv_id, ax in zip(dv_ids, axs):
                 dv = ranks_dvs[dv_id]

                 sns.barplot(x=iv['label'], y=dv['label'], palette=iv['palette'],
                             data=ranks, ax=ax, estimator=estimator)
                 ax.set(ylim=(0, dv['scale'][-1]))
                 ax.yaxis.set_major_locator(ticker.MultipleLocator(1)) # Fix the axis ticks

             return (fig, axs)
```

```
In [30]: def rank_samples(iv_id, dv_id):
             """Returns the list of ranks (DV) for each IV value"""
             samples = []
             iv = trials_ivs[iv_id]
             for iv_value in iv['categorical']:
                 dv_label = ranks_dvs[dv_id]['label']
```

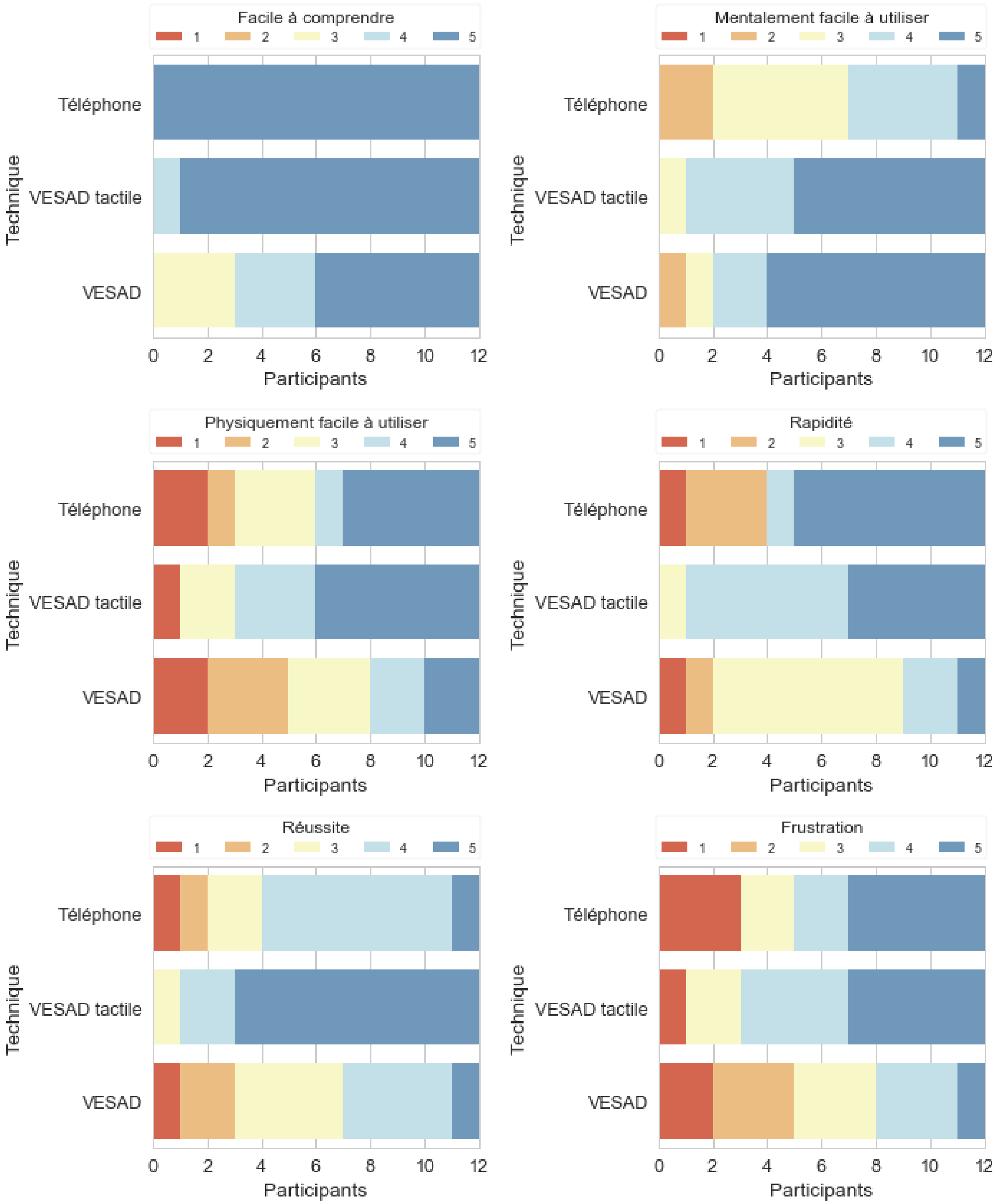
```
            sample = ranks[ranks[iv['label']] == iv_value][dv_label]\
                      .reset_index(drop=True)
            samples.append(sample)
        return samples
```

```
In [31]: def test_ranks(iv_id, dv_ids, **args):
             results = []
             iv = trials_ivs[iv_id]

             for dv_id in dv_ids:
                 dv = ranks_dvs[dv_id]
                 samples = rank_samples(iv_id, dv_id)
                 H, p = stats.kruskal(*samples)
                 results.append([iv['label'], dv['label'], H, p])

             results = pd.DataFrame(results, columns=[labels['iv'], labels['dv'],
                                                      'Kruskal-Wallis H',
                                                      labels['p_value']])
             p_values_correction(results, **args)
             return results
```

```
In [32]: def test_pairwise_ranks(iv_id, dv_ids, **args):
             iv = trials_ivs[iv_id]
             iv_category_ids = range(len(iv['categorical']))

             results = []
             for dv_id in dv_ids:
                 dv = ranks_dvs[dv_id]
                 samples = rank_samples(iv_id, dv_id)
                 sample_pairs = itertools.combinations(iv_category_ids, 2)
                 for id1, id2 in sample_pairs:
                     U, p = stats.mannwhitneyu(samples[id1], samples[id2])
                     mean_diff, mean_diff_per = mean_difference(samples[id1], samples[id2])
                     results.append([iv['label'], iv['categorical'][id1],
                                     iv['categorical'][id2], dv['label'],
                                     mean_diff, mean_diff_per, U, p])

             columns = [labels['iv'], labels['iv_value'] + ' 1',
                        labels['iv_value'] + ' 2', labels['dv'],
                        labels['mean_difference'], labels['mean_difference_percentage'],
                        'Mann-Whitney U', labels['p_value']]
             results = pd.DataFrame(results, columns=columns)
             p_values_correction(results, **args)
             return results
```

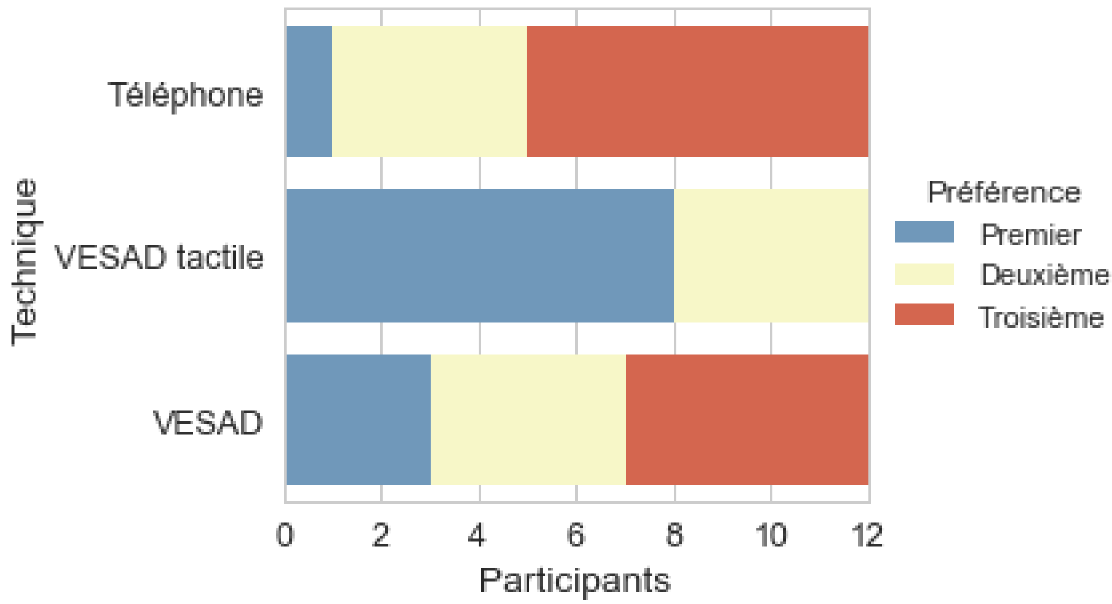
We display the rank and preference distributions:

```
In [33]: (fig, axs) = plot_ranks_distributions('technique', ranks_dvs.columns[0:-1])
         fig.savefig('ranks_distributions.png')
```




```
In [34]: (fig, axs) = plot_ranks_distributions('technique', ['preference'])

         ax_handles, ax_labels = axs[0].get_legend_handles_labels()
         legend = axs[0].legend(ax_handles[::-1], labels['preferences'],
                                loc='center left', bbox_to_anchor=(1, 0.5),
                                title=ranks_dvs['preference']['label'])
```

```
fix_legend_fontsize(legend)

fig.savefig('preferences_distribution.png')
```



We use the Kruskal-Wallis test (Benjamini–Hochberg correction) on each question to check if there is statistical significative differences between the ranks among TECHNIQUE:

```
In [35]: test_ranks('technique', ranks_dvs, correction_method='fdr_bh')
```

```
Out[35]:   Variable indépendante              Variable dépendante  Kruskal-Wallis H  \
         0             Technique               Facile à comprendre         11.001420
         1             Technique   Mentalement facile à utiliser         10.905742
         2             Technique  Physiquement facile à utiliser          4.148121
         3             Technique                          Rapidité          7.130846
         4             Technique                          Réussite         13.784269
         5             Technique                       Frustration          4.500057
         6             Technique                        Préférence         12.638889

            Valeur p
         0  0.007497
         1  0.007497
         2  0.125674
         3  0.039599
         4  0.006303
         5  0.122962
         6  0.006303
```

All the questions, except Physically Easy to Use and Frustration, are statistically significants: Easy to Understand ($p = 0.007$), Mentally Easy to Use ($p = 0.007$), Subjective Speed ($p = 0.04$), Subjective Performance ($p = 0.006$), Preference ($p = 0.006$).

We use then pairwise Mann-Whitney tests (Benjamini–Hochberg correction) for the significant questions above:

```
In [36]: test_pairwise_ranks('technique', ['easy_understand', 'mentally_easy_use',
                                           'could_go_fast', 'subjective_performance',
                                           'preference'], correction_method='fdr_bh')
```

```
Out[36]:    Variable indépendante    Valeur VI 1    Valeur VI 2  \
        0               Technique      Téléphone  VESAD tactile
        1               Technique      Téléphone          VESAD
        2               Technique  VESAD tactile          VESAD
        3               Technique      Téléphone  VESAD tactile
        4               Technique      Téléphone          VESAD
        5               Technique  VESAD tactile          VESAD
        6               Technique      Téléphone  VESAD tactile
        7               Technique      Téléphone          VESAD
        8               Technique  VESAD tactile          VESAD
        9               Technique      Téléphone  VESAD tactile
        10              Technique      Téléphone          VESAD
        11              Technique  VESAD tactile          VESAD
        12              Technique      Téléphone  VESAD tactile
        13              Technique      Téléphone          VESAD
        14              Technique  VESAD tactile          VESAD

                      Variable dépendante  Différence des moyennes  \
        0             Facile à comprendre                 0.083333
        1             Facile à comprendre                 0.750000
        2             Facile à comprendre                 0.666667
        3   Mentalement facile à utiliser                -1.166667
        4   Mentalement facile à utiliser                -1.083333
        5   Mentalement facile à utiliser                 0.083333
        6                        Rapidité                -0.500000
        7                        Rapidité                 0.750000
        8                        Rapidité                 1.250000
        9                        Réussite                -1.166667
        10                       Réussite                 0.333333
        11                       Réussite                 1.500000
        12                     Préférence                 1.166667
        13                     Préférence                 0.333333
        14                     Préférence                -0.833333

            Différence des moyennes (%)  Mann-Whitney U  Valeur p
        0                      1.694915            66.0  0.215732
        1                     17.647059            36.0  0.008605
        2                     15.686275            40.5  0.020977
        3                    -25.925926            23.0  0.005031
        4                    -24.528302            28.5  0.010151
        5                      1.886792            69.5  0.475027
        6                    -11.538462            70.5  0.475027
        7                     24.324324            48.5  0.126838
        8                     40.540541            21.0  0.004655
        9                    -25.000000            22.5  0.004655
        10                    10.526316            57.0  0.215732
        11                    47.368421            17.5  0.003904
```

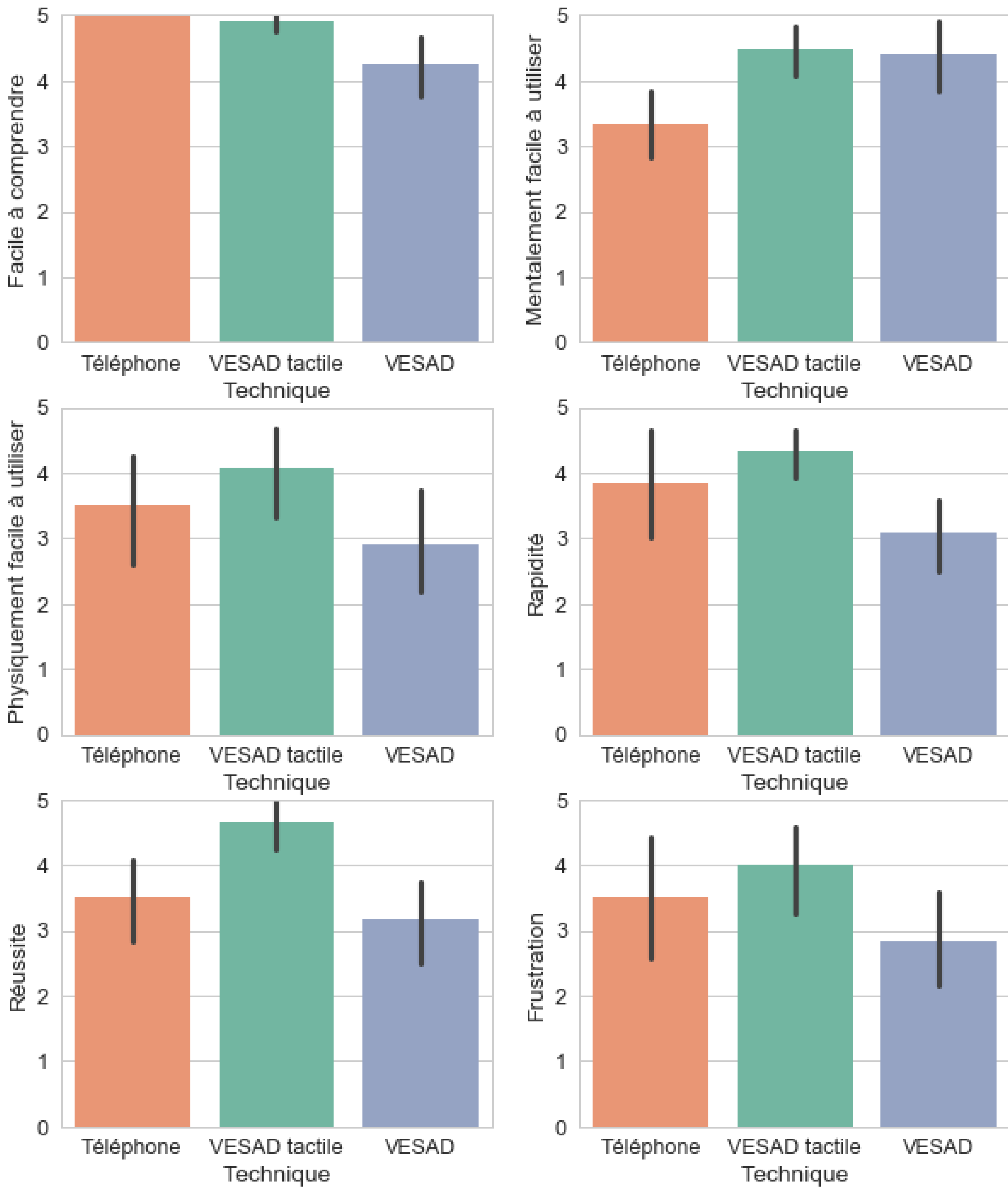
```
12                    87.500000              16.0  0.003904
13                    15.384615              56.0  0.215732
14                   -38.461538              32.0  0.013011
```

We display the mean and 95% CI of each question:

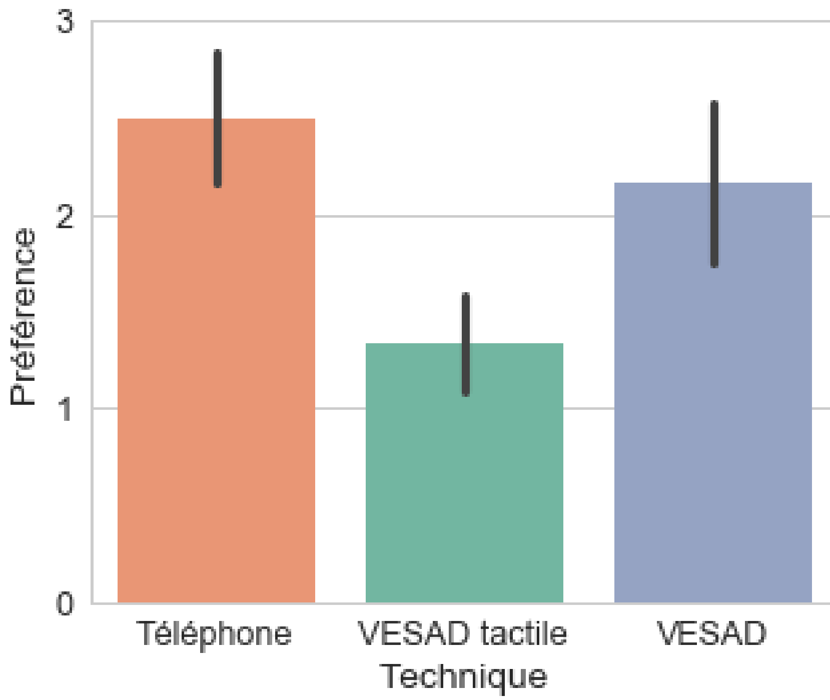
```
In [37]: ranks.groupby([trials_ivs['technique']['label']])\
              .aggregate(print_mean_ci).loc[:, ranks_dvs.loc['label', :]].transpose()
```

```
Out[37]: Technique                                  Téléphone      VESAD tactile  \
         Facile à comprendre              5.00 [5.00, 5.00]  4.92 [4.75, 5.00]
         Mentalement facile à utiliser    3.33 [2.83, 3.83]  4.50 [4.08, 4.83]
         Physiquement facile à utiliser   3.50 [2.67, 4.33]  4.08 [3.42, 4.67]
         Rapidité                         3.83 [2.92, 4.67]  4.33 [4.00, 4.67]
         Réussite                         3.50 [2.83, 4.00]  4.67 [4.25, 5.00]
         Frustration                      3.50 [2.67, 4.33]  4.00 [3.33, 4.58]
         Préférence                       2.50 [2.17, 2.83]  1.33 [1.08, 1.58]

         Technique                                      VESAD
         Facile à comprendre              4.25 [3.75, 4.67]
         Mentalement facile à utiliser    4.42 [3.83, 4.92]
         Physiquement facile à utiliser   2.92 [2.17, 3.67]
         Rapidité                         3.08 [2.50, 3.58]
         Réussite                         3.17 [2.58, 3.75]
         Frustration                      2.83 [2.17, 3.42]
         Préférence                       2.17 [1.75, 2.58]
```

```
In [38]: (fig, axs) = plot_ranks('technique', ranks_dvs.columns[0:-1])
         fig.savefig('ranks.png')
```




```
In [39]: (fig, axs) = plot_ranks('technique', ['preference'])
         fig.savefig('preferences.png')
```



Overall significant results are:

- **Easy to Understand**: *PhoneOnly* is significantly better than *MidAirInArOut* ($p = 0.009$), and seems a little better than *PhoneInArOut*.
- **Physically Easy to Use**: There is no significant differences due to TECHNIQUE; they seem scored similar.
- **Mentally Easy to Use**: *PhoneOnly* is statistically and practically worst than *PhoneInArOut* ($p = 0.005$) and *MidAirInArOut* ($p = 0.01$).
- **Subjective Speed**: *PhoneInArOut* is significantly better than *MidAirInArOut* ($p = 0.05$).
- **Subjective Performance**: *PhoneInArOut* is statistically better than *PhoneOnly* ($p = 0.005$) and *MidAirInArOut* ($p = 0.004$).
- **Frustration**: There is no significant differences due to TECHNIQUE; they seem scored similar.
- **Preference**: *PhoneInArOut* is significantly prefered to *PhoneOnly* ($p = 0.004$) and *MidAirInArOut* ($p = 0.01$).

0.3 3. Participant trials

Some functions for the analysis:

```
In [40]: def trial_samples(iv_id, dv_id, data=trials):
             iv, dv = trials_ivs[iv_id], trials_dvs[dv_id]
             return [data[data[iv['label']] == iv_value][dv]\
                     for iv_value in iv['categorical']]
```

```
In [41]: def trial_means(iv_ids, dv_ids, data=trials):
             iv_labels = [trials_ivs.at['label', iv_id] for iv_id in iv_ids]
             dv_labels = [trials_dvs[dv_id] for dv_id in dv_ids]
             return data.groupby(iv_labels).aggregate(print_mean_ci).loc[:, dv_labels]
```

In [42]:
```
def melt_trials(value_vars, var_name, value_name, data=trials):
    return pd.melt(data, id_vars=trials_ivs.loc['label', :],
                   value_vars=value_vars, var_name=var_name,
                   value_name=value_name)
```

In [43]:
```
def filter_trials_outliers(dv_id, data=trials, nth_percentile_trimed=5):
    dv = trials_dvs[dv_id]
    pmin, pmax = np.nanpercentile(data[dv], [nth_percentile_trimed,
                                             100 - nth_percentile_trimed])
    return data[(np.isnan(data[dv]) == False) & (pmin < data[dv])\
                & (data[dv] < pmax)]
```

In [44]:
```
def test_pairwise_trials(iv_id, dv_id, data=trials, **args):
    results = []
    iv, dv = trials_ivs[iv_id], trials_dvs[dv_id]
    iv_category_ids = range(len(iv['categorical']))

    samples = trial_samples(iv_id, dv_id, data)
    sample_pairs = itertools.combinations(iv_category_ids, 2)
    for id1, id2 in sample_pairs:
        T, p = stats.ttest_ind(samples[id1], samples[id2])
        mean_diff, mean_diff_per = mean_difference(samples[id1], samples[id2])
        results.append([iv['label'], iv['categorical'][id1],
                        iv['categorical'][id2], dv, mean_diff,
                        mean_diff_per, T, p])

    columns = [labels['iv'], labels['iv_value'] + ' 1',
               labels['iv_value'] + ' 2', labels['dv'],
               labels['mean_difference'], labels['mean_difference_percentage'],
               labels['t_statistic'], labels['p_value']]
    results = pd.DataFrame(results, columns=columns)

    p_values_correction(results, **args)
    return results
```

In [45]:
```
def test_non_normal_trials(iv_ids, dv_ids, data=trials, **args):
    results = []

    for dv_id in dv_ids:
        for iv_id in iv_ids:
            samples = trial_samples(iv_id, dv_id, data)
            H, p = stats.kruskal(*samples)
            results.append([trials_ivs.at['label', iv_id], trials_dvs[dv_id],
                            H, p])

    results = pd.DataFrame(results, columns=[labels['iv'], labels['dv'],
                                             'Kruskal-Wallis H',
                                             labels['p_value']])
    p_values_correction(results, **args)
    return results
```

```
In [46]: def test_pairwise_non_normal_trials(iv_ids, dv_ids, data=trials, **args):
             results = []

             for dv_id in dv_ids:
                 for iv_id in iv_ids:
                     iv, dv = trials_ivs[iv_id], trials_dvs[dv_id]
                     samples = trial_samples(iv_id, dv_id, data)

                     iv_category_ids = range(len(iv['categorical']))
                     sample_pairs = itertools.combinations(iv_category_ids, 2)
                     for id1, id2 in sample_pairs:
                         U, p = stats.mannwhitneyu(samples[id1], samples[id2])
                         mean_diff, mean_diff_per = mean_difference(samples[id1],
                                                                    samples[id2])
                         results.append([iv['label'], iv['categorical'][id1],
                                         iv['categorical'][id2], dv, mean_diff,
                                         mean_diff_per, U, p])

             columns = [labels['iv'], labels['iv_value'] + ' 1',
                        labels['iv_value'] + ' 2', labels['dv'],
                        labels['mean_difference'], labels['mean_difference_percentage'],
                        'Mann-Whitney U', labels['p_value']]
             results = pd.DataFrame(results, columns=columns)
             p_values_correction(results, **args)
             return results
```

```
In [47]: def plot_trials(iv_ids_list, dv_id, data=trials, kind='bar', **args):
             dv = trials_dvs[dv_id]
             if (len(iv_ids_list) == 0):
                 iv_ids_list = [[iv_id] for id_id in trials_ivs.columns]

             fig, axs = subplots(len(iv_ids_list))
             for id_ids, ax in zip(iv_ids_list, axs):

                 ivs = [trials_ivs[id_id] for id_id in id_ids]
                 if (len(ivs) == 1):
                     iv = ivs[0]

                     if (kind == 'bar'):
                         sns.barplot(x=iv['label'], y=dv, data=data,
                                     palette=iv['palette'], ax=ax, **args)

                     elif (kind == 'box'):
                         sns.boxplot(x=iv['label'], y=dv, data=data,
                                     palette=iv['palette'], ax=ax, **args)

                     elif (kind == 'count'):
                         sns.countplot(hue=iv['label'], x=dv, data=data,
                                       palette=iv['palette'], ax=ax, **args)
                         ax.set(ylabel='Count')
                         ax.legend(loc='upper right', title=labels['count'],
```

```
                                  frameon=True)

            elif (len(ivs) == 2):
                if (kind == 'bar'):
                    sns.barplot(x=ivs[1]['label'], y=dv, hue=ivs[0]['label'],
                                data=data, palette=ivs[0]['palette'], ax=ax, **args)
                ax.legend(frameon=True, loc='upper left', bbox_to_anchor=(1, 1))

        return (fig, axs)
```

0.3.1 3.1. Task completion time

We perform a full factorial ANOVA with the model: TCT ~ TECHNIQUE x TEXT_SIZE x DISTANCE + TECHNIQUE x ORDERING.

We should test all the assumptions of an ANOVA : independence of measure points, normality, homogeneity of variance. The trials are independent from each other, since they are generated randomly. The data is non-normal, but the ANOVA can tolerate non-normal data with skewed distribution. The homogeneity of variance is less important when the sample sizes are equal.

```
In [48]: tct_model = ols('total_time ~ technique * text_size * distance'
                         + '+ technique * ordering', data=trials_for_anova).fit()
         sm.stats.anova_lm(tct_model, typ=2)
```

Out[48]:

	sum_sq	df	F	PR(>F)
technique	256284.863291	2.0	52.727496	4.664381e-20
text_size	6524.396184	1.0	2.684630	1.024855e-01
distance	273.284406	1.0	0.112450	7.376324e-01
ordering	12606.272262	2.0	2.593587	7.661302e-02
technique:text_size	10779.057863	2.0	2.217660	1.108428e-01
technique:distance	1129.444917	2.0	0.232370	7.928115e-01
text_size:distance	10418.141005	1.0	4.286812	3.935962e-02
technique:ordering	47629.511187	4.0	4.899597	7.897650e-04
technique:text_size:distance	17350.562563	2.0	3.569667	2.950225e-02
Residual	656174.846745	270.0	NaN	NaN

The main significant effect on TCT is TECHNIQUE ($F = 52.7$, $p < 0.0001$). There is also interaction effects: TEXT_SIZE x DISTANCE ($F = 4.3$, $p = 0.04$), TECHNIQUE x ORDERING ($F = 4.9$, $p = 0.0007$) and TECHNIQUE x TEXT_SIZE x DISTANCE ($F = 3.0$, $p = 0.03$).

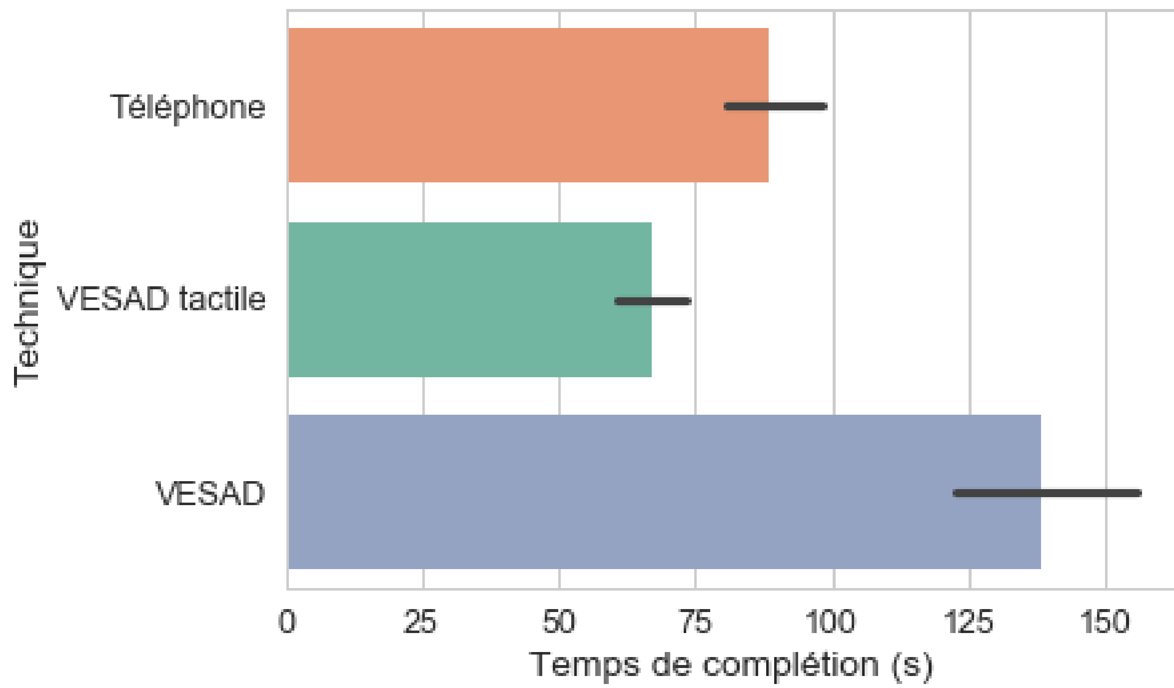
We display mean TCT values with 95% CI for these conditions:

```
In [49]: trial_means(['technique'], ['total_time'])
```

Out[49]:

Technique	Temps de complétion (s)
Téléphone	88.70 [79.44, 97.51]
VESAD tactile	66.86 [60.82, 73.78]
VESAD	138.17 [121.86, 156.24]

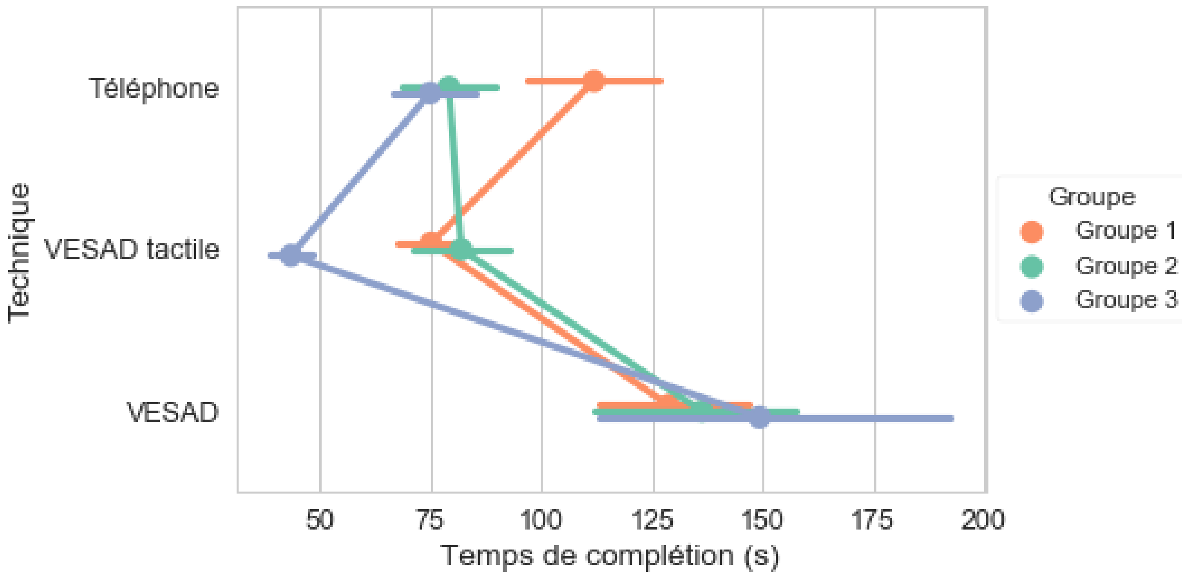
```
In [50]: ax = sns.barplot(x=trials_dvs['total_time'], y=technique['label'],
                          palette=technique['palette'], data=trials)
         ax.get_figure().savefig('tct.png')
```




```
In [51]: trial_means(['ordering', 'technique'], ['total_time']).unstack()
```

```
Out[51]:           Temps de complétion (s)                          \
         Technique                Téléphone          VESAD tactile
         Groupe
         Groupe 1   111.98 [95.69, 129.01]  75.15 [67.83, 82.57]
         Groupe 2     79.19 [69.86, 90.43]  81.97 [71.06, 92.69]
         Groupe 3     74.93 [66.13, 85.17]  43.46 [38.51, 48.20]


         Technique                       VESAD
         Groupe
         Groupe 1   128.82 [111.16, 146.32]
         Groupe 2   136.32 [112.82, 157.68]
         Groupe 3   149.36 [114.44, 190.34]
```

```
In [52]: ax = sns.pointplot(x=trials_dvs['total_time'], y=technique['label'],
                            hue=ordering['label'], palette=ordering['palette'],
                            data=trials, dodge=True)
         config_legend(ax, 'ordering')
         ax.get_figure().savefig('tct_ordering.png')
```

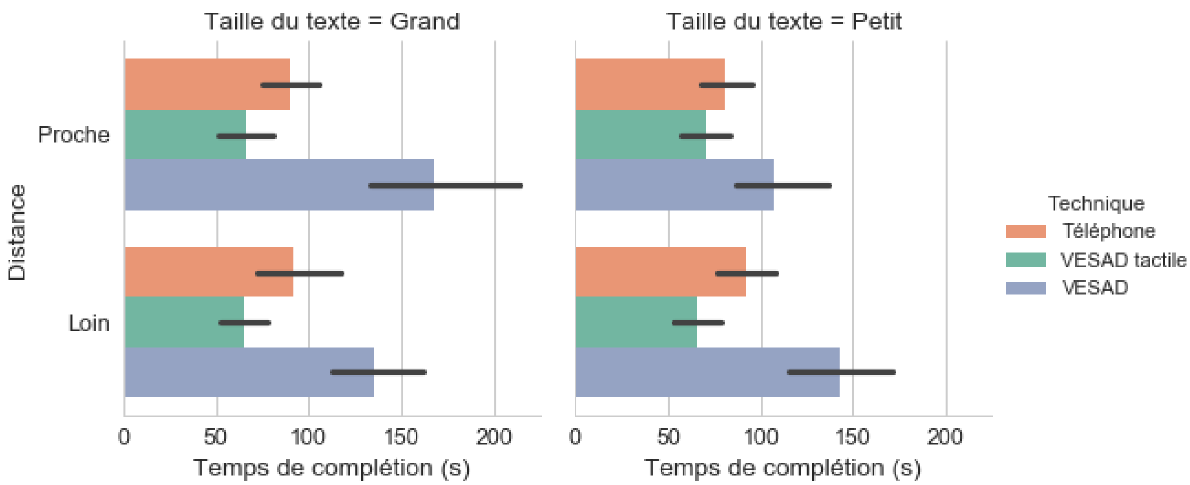


It seems that participants who started with *PhoneOnly* were slower with this technique than the the other groups. Similarly, participants who finished with *PhoneInArOut* were faster with this technique. It indicates there is a learning curve on the task, but interestingly participants from all groups performed equally with *MidAirInArOut* technique.

```
In [53]: trial_means(['text_size', 'distance', 'technique'], ['total_time']).unstack()
```

```
Out[53]:                          Temps de complétion (s)                          \
         Technique                              Téléphone         VESAD tactile
         Taille du texte Distance
         Grand           Proche     89.98 [73.86, 105.05]  65.81 [52.90, 80.86]
                         Loin       92.24 [71.66, 114.40]  65.35 [52.55, 77.78]
         Petit           Proche      80.44 [67.00, 94.63]  70.61 [58.01, 83.72]
                         Loin       92.13 [76.62, 110.48]  65.67 [53.22, 79.23]


         Technique                                   VESAD
         Taille du texte Distance
         Grand           Proche    167.33 [131.76, 216.83]
                         Loin      135.29 [111.80, 162.08]
         Petit           Proche     107.43 [84.19, 136.76]
                         Loin      142.61 [115.30, 169.66]
```

```
In [54]: g = sns.factorplot(x=trials_dvs['total_time'], y=distance['label'],
                            col=text_size['label'], hue=technique['label'],
                            data=trials, palette=technique['palette'], kind='bar')
         g.savefig('tct_2.png')
```



Interactions on TCT due to TEXT_SIZE x DISTANCE don't seem importants.

We compare the TCT for the three techniques only with pairwise t-tests (Benjamini–Hochberg correction) .

```
In [55]: test_pairwise_trials('technique', 'total_time', correction_method='fdr_bh')
```

```
Out[55]:   Variable indépendante    Valeur VI 1    Valeur VI 2  \
        0             Technique      Téléphone  VESAD tactile
        1             Technique      Téléphone          VESAD
        2             Technique  VESAD tactile          VESAD

              Variable dépendante  Différence des moyennes  \
        0  Temps de complétion (s)                21.840362
        1  Temps de complétion (s)               -49.467731
        2  Temps de complétion (s)               -71.308093

           Différence des moyennes (%)  Statistique T      Valeur p
        0                    32.666819       3.891218  1.862003e-04
        1                   -35.803106      -5.143480  2.201995e-06
        2                   -51.610437      -7.746559  3.247150e-11
```

Results:

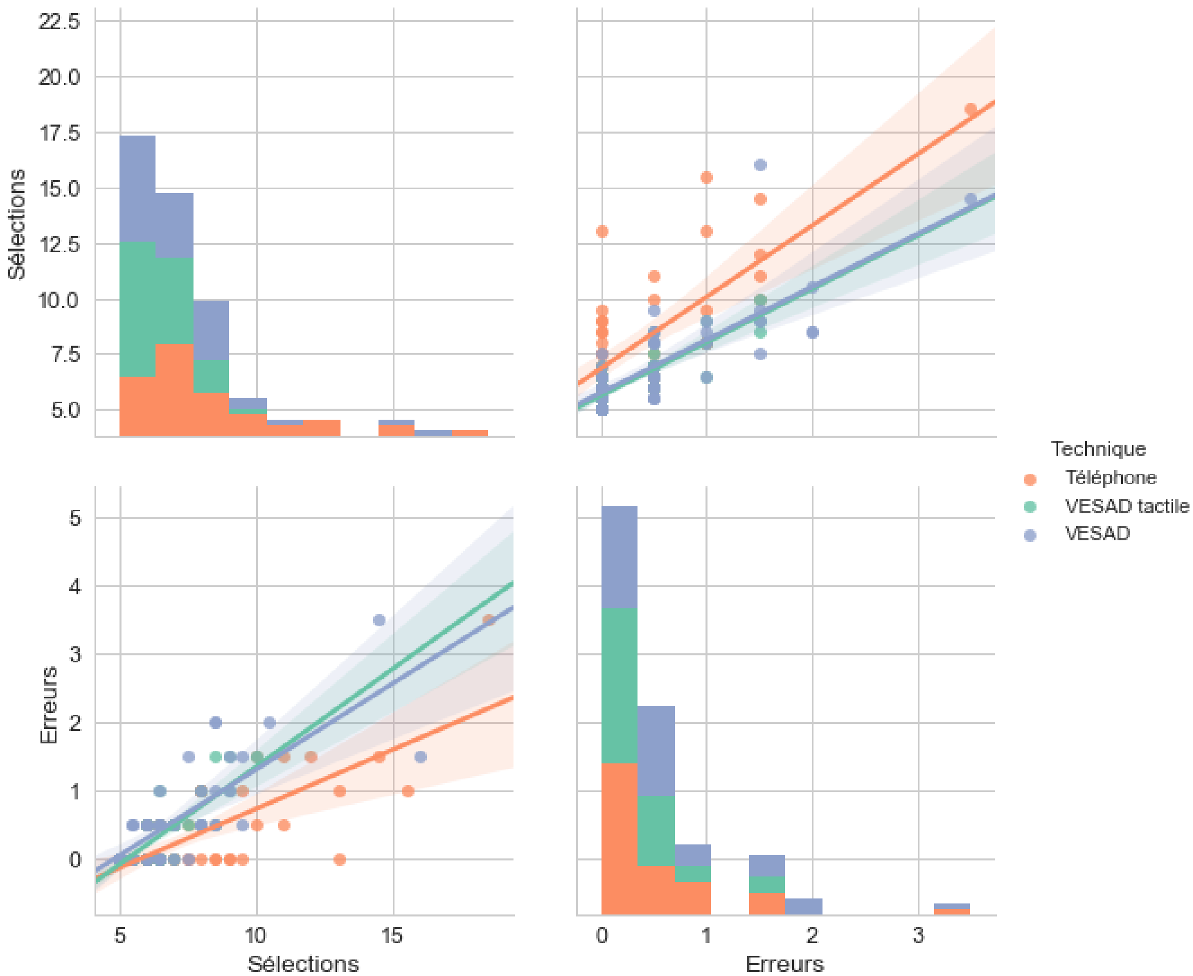
- *PhoneInArOut* is 71s (+52%) faster than *MidAirInArOut* ($p < 0.0001$).
- *PhoneOnly* is 49s (+36%) faster than *MidAirInArOut* ($p < 0.0001$).
- *PhoneInArOut* is 22s (+33%) faster than *PhoneOnly* ($p = 0.0004$).

Differences due to interactions don't seem importants.

0.3.2 3.2. Errors

We visualize first the SELECTIONS and ERRORS distributions:

```
In [56]: g = sns.pairplot(trials, hue=technique['label'], kind='reg',
                          vars=[trials_dvs['selections_count'], trials_dvs['errors']],
                          palette=technique['palette'], size=4);
         g.savefig('selections_errors_distributions.png')
```



It seems that a user makes as much errors as she makes selections. The relation is almost the same for each technique, even if users seems to make more selections for the same number of errors with PhoneOnly.

We can't use ANOVA on SELECTIONS and ERRORS variables as their distributions are exponentials. We use instead the Kruskal-Wallis test (Benjamini–Hochberg correction) to check if there is significative differences due to TECHNIQUE, TEXT_SIZE, DISTANCE or ORDERING.

```
In [57]: test_non_normal_trials(['technique', 'text_size', 'distance', 'ordering'],
                                ['selections_count', 'errors'])
```

Out[57]:

	Variable indépendante	Variable dépendante	Kruskal-Wallis H	Valeur p
0	Technique	Sélections	15.125918	0.004155
1	Taille du texte	Sélections	0.003897	0.950224
2	Distance	Sélections	0.052701	0.935347
3	Groupe	Sélections	11.510535	0.010095
4	Technique	Erreurs	5.201541	0.148433
5	Taille du texte	Erreurs	0.103661	0.935347
6	Distance	Erreurs	0.201599	0.935347
7	Groupe	Erreurs	11.153096	0.010095

Only TECHNIQUE ($p = 0.004$) and ORDERING ($p = 0.01$) have a significant effect on SELECTIONS. But, only ORDERING ($p = 0.01$) have a significant effect on ERRORS.

We use then pairwise Mann-Whitney tests (Benjamini–Hochberg correction) for the significant questions above:

```
In [58]: test_pairwise_non_normal_trials(['technique', 'ordering'],
                                         ['selections_count', 'errors'])
```

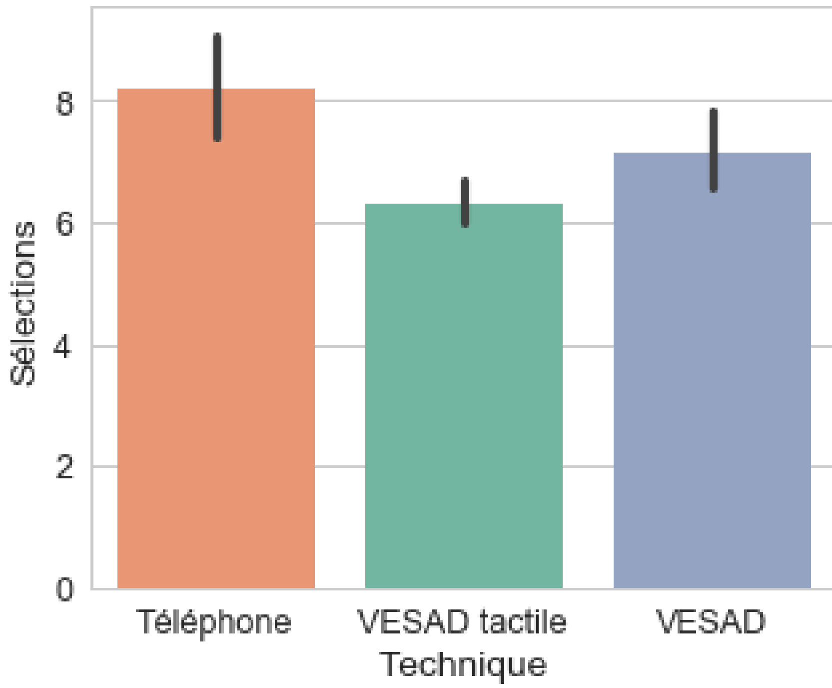
```
Out[58]:    Variable indépendante    Valeur VI 1    Valeur VI 2 Variable dépendante  \
        0               Technique      Téléphone  VESAD tactile          Sélections
        1               Technique      Téléphone          VESAD          Sélections
        2               Technique  VESAD tactile          VESAD          Sélections
        3                  Groupe       Groupe 1       Groupe 2          Sélections
        4                  Groupe       Groupe 1       Groupe 3          Sélections
        5                  Groupe       Groupe 2       Groupe 3          Sélections
        6               Technique      Téléphone  VESAD tactile             Erreurs
        7               Technique      Téléphone          VESAD             Erreurs
        8               Technique  VESAD tactile          VESAD             Erreurs
        9                  Groupe       Groupe 1       Groupe 2             Erreurs
        10                 Groupe       Groupe 1       Groupe 3             Erreurs
        11                 Groupe       Groupe 2       Groupe 3             Erreurs

            Différence des moyennes  Différence des moyennes (%)  Mann-Whitney U  \
        0                  1.864583                    29.537954           627.0
        1                  1.041667                    14.598540           864.5
        2                 -0.822917                   -11.532847           915.0
        3                  1.729167                    26.265823           755.0
        4                  1.583333                    23.529412           755.0
        5                 -0.145833                    -2.167183          1136.0
        6                  0.125000                    42.857143          1083.5
        7                 -0.166667                   -28.571429           957.0
        8                 -0.291667                   -50.000000           873.0
        9                  0.333333                   103.225806           846.5
        10                 0.343750                   110.000000           761.5
        11                 0.010417                     3.333333          1039.0

            Valeur p
        0   0.000635
        1   0.029356
        2   0.059881
        3   0.005196
        4   0.005196
        5   0.454285
        6   0.312007
        7   0.083050
        8   0.026306
        9   0.020183
        10  0.005196
        11  0.205336
```

```
In [59]: trial_means(['technique'], ['selections_count', 'errors'])
```

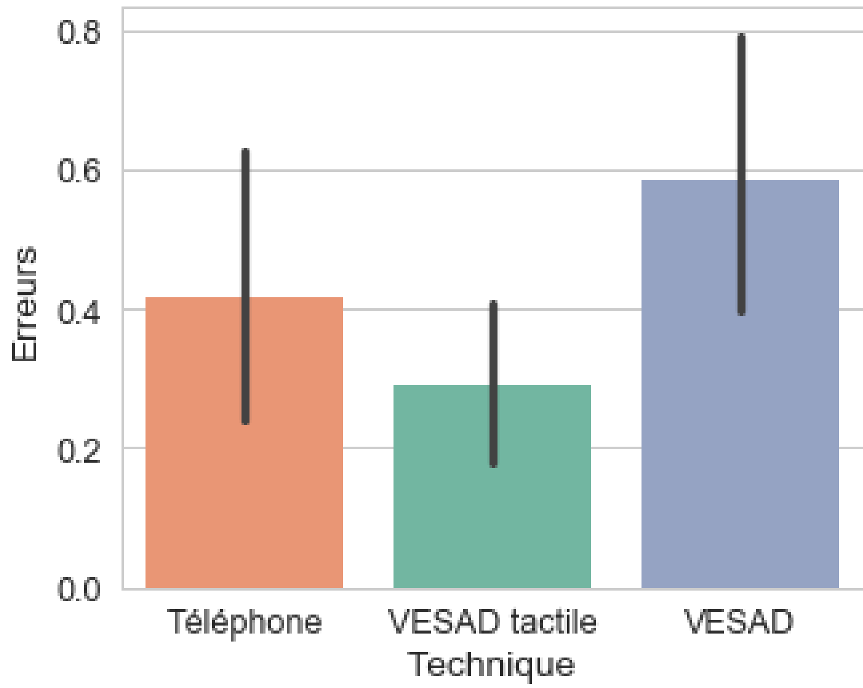
```
Out[59]:                          Sélections            Erreurs
        Technique
```

```
Téléphone      8.18 [7.46, 9.03]  0.42 [0.24, 0.61]
VESAD tactile  6.31 [5.98, 6.68]  0.29 [0.18, 0.41]
VESAD          7.14 [6.54, 7.83]  0.58 [0.39, 0.81]
```

In [60]:
```
(fig, axs) = plot_trials([['technique']], 'selections_count')
fig.savefig('selections.png')
```



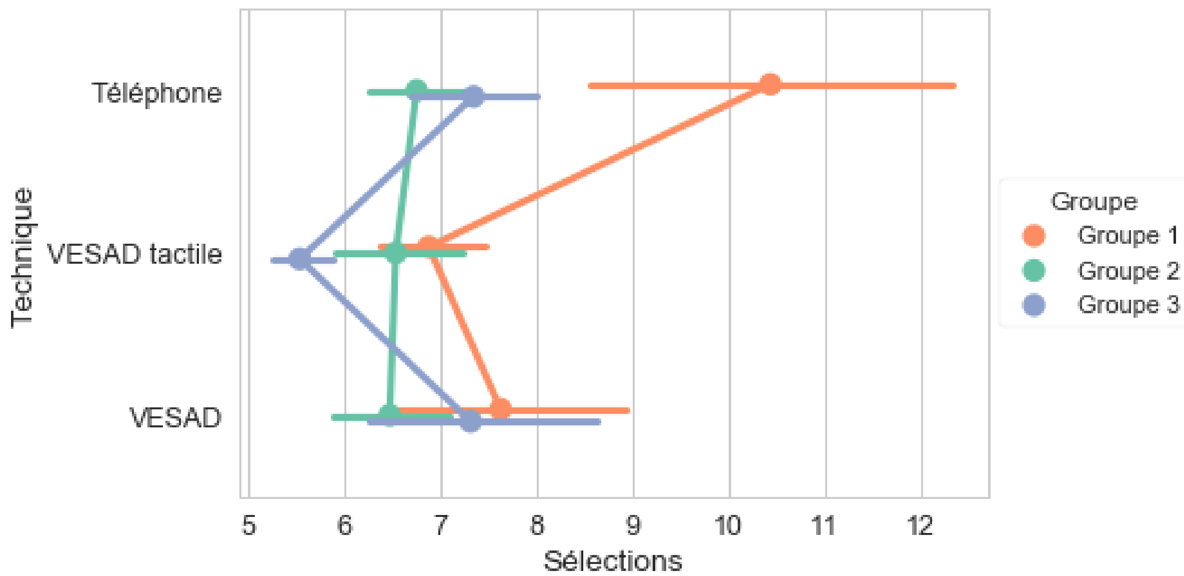
In [61]:
```
(fig, axs) = plot_trials([['technique']], 'errors')
fig.savefig('errors.png')
```



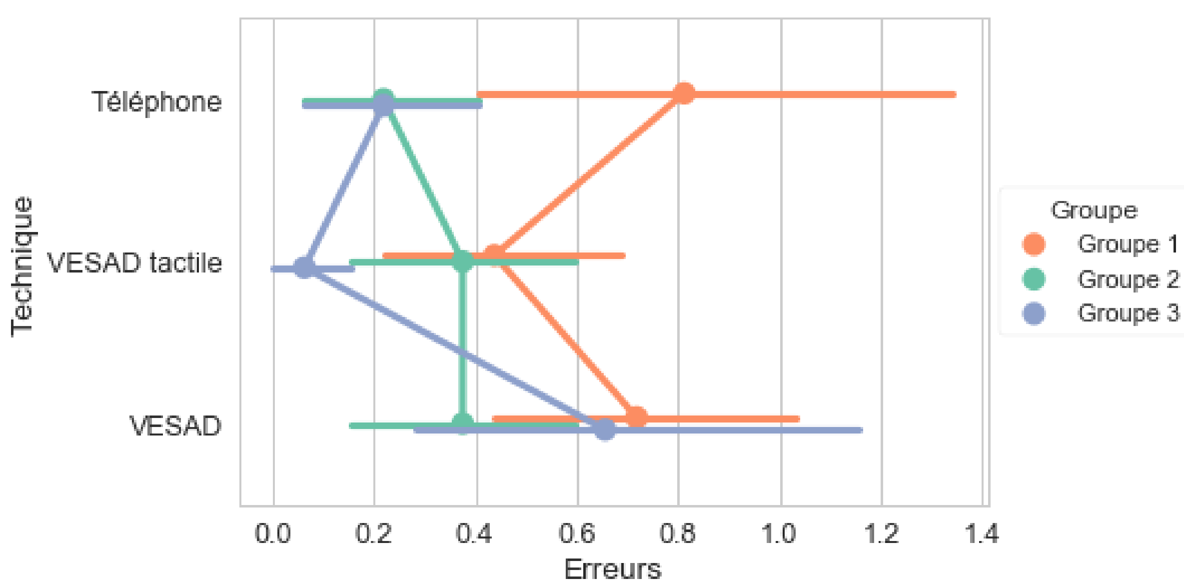

```
In [62]: trial_means(['ordering', 'technique'], ['selections_count', 'errors'])
```

```
Out[62]:                                Sélections            Erreurs
         Groupe   Technique
         Groupe 1 Téléphone      10.44 [8.75, 12.34]  0.81 [0.44, 1.31]
                  VESAD tactile    6.88 [6.38, 7.47]  0.44 [0.22, 0.69]
                  VESAD            7.62 [6.50, 9.12]  0.72 [0.41, 1.03]
         Groupe 2 Téléphone        6.75 [6.25, 7.31]  0.22 [0.06, 0.41]
                  VESAD tactile    6.53 [5.88, 7.25]  0.38 [0.16, 0.62]
                  VESAD            6.47 [5.84, 7.19]  0.38 [0.16, 0.62]
         Groupe 3 Téléphone        7.34 [6.75, 8.06]  0.22 [0.06, 0.41]
                  VESAD tactile    5.53 [5.25, 5.88]  0.06 [0.00, 0.16]
                  VESAD            7.31 [6.22, 8.62]  0.66 [0.25, 1.19]
```

```
In [63]: ax = sns.pointplot(x=trials_dvs['selections_count'], y=technique['label'],
                            hue=ordering['label'], palette=ordering['palette'],
                            data=trials, dodge=True)
         config_legend(ax, 'ordering')
         ax.get_figure().savefig('selections_ordering.png')
```




```
In [64]: ax = sns.pointplot(x=trials_dvs['errors'], y=technique['label'],
                            hue=ordering['label'], palette=ordering['palette'],
                            data=trials, dodge=True)
         config_legend(ax, 'ordering')
         ax.get_figure().savefig('errors_ordering.png')
```



0.3.3 3.3. Navigation

Variable meanings:

- Selection Time = time spent looking for where to drop an item that had been picked

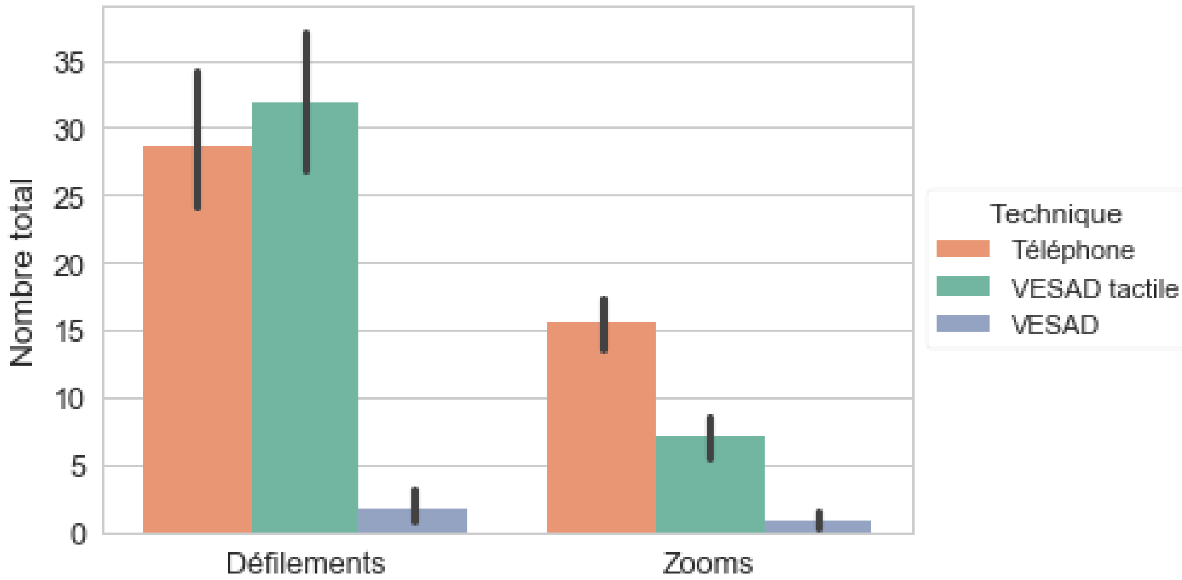
- Selection Distance = distance travelled by the finger with an item selected
- Head Phone Distance = sum of the distance between the head and the phone

In [65]:
```
# Data preparation
trial_counts = melt_trials(var_name=labels['category'],
                           value_name=labels['count'],
                           value_vars=[trials_dvs['pan_count'],
                                       trials_dvs['zoom_count']])

trial_times = melt_trials(var_name=labels['category'],
                          value_name=labels['time'],
                          value_vars=[trials_dvs['selections_time'],
                                      trials_dvs['pan_time'],
                                      trials_dvs['zoom_time']])

trial_distances_dvs = [trials_dvs['selections_projected_distance'],
                       trials_dvs['pan_projected_distance'],
                       trials_dvs['zoom_projected_distance'],
                       trials_dvs['absolute_head_phone_distance']]
trial_distances = melt_trials(var_name=labels['category'],
                              value_name=labels['distance'],
                              value_vars=trial_distances_dvs)
```

In [66]:
```
trial_means(['technique'], ['pan_count', 'zoom_count'])
```

Out[66]:
```
                        Défilements                 Zooms
Technique
Téléphone      28.70 [24.02, 33.95]  15.53 [13.68, 17.50]
VESAD tactile  31.81 [26.34, 36.75]     7.14 [5.71, 8.78]
VESAD             1.83 [0.82, 3.04]     0.83 [0.29, 1.52]
```

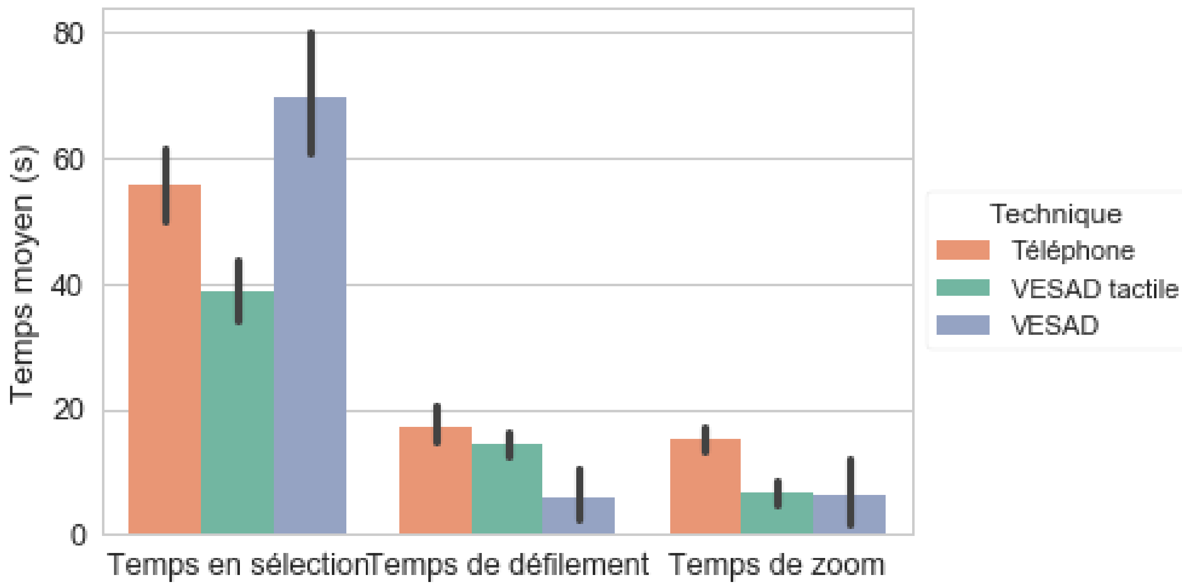
In [67]:
```
ax = sns.barplot(x=labels['category'], y=labels['count'],
                 hue=technique['label'], palette=technique['palette'],
                 data=trial_counts)
config_legend(ax, 'technique')
ax.set(xlabel='')
ax.get_figure().savefig('navigation_count.png')
```




```
In [68]: trial_means(['technique'], ['selections_time', 'pan_time', 'zoom_time'])
```

```
Out[68]:                  Temps en sélection   Temps de défilement  \
         Technique
         Téléphone      55.74 [49.67, 62.01]  17.12 [14.76, 19.92]
         VESAD tactile  38.98 [34.49, 43.86]  14.53 [12.59, 16.69]
         VESAD          69.68 [60.93, 79.01]    5.97 [2.43, 10.61]

                              Temps de zoom
         Technique
         Téléphone      15.18 [13.08, 17.17]
         VESAD tactile     6.70 [4.99, 8.57]
         VESAD            6.24 [1.57, 12.36]
```

```
In [69]: ax = sns.barplot(x=labels['category'], y=labels['time'], data=trial_times,
                          hue=technique['label'], palette=technique['palette'])
         config_legend(ax, 'technique')
         ax.set(xlabel='')
         ax.get_figure().savefig('navigation_time.png')
```




```
In [70]: trial_means(['technique'], ['selections_projected_distance',
                                     'pan_projected_distance',
                                     'zoom_projected_distance',
                                     'absolute_head_phone_distance'])
```

```
Out[70]:               Distance en sélection Distance de défilement   Distance de zoom  \
         Technique
         Téléphone         4.87 [4.03, 5.72]      1.50 [1.20, 1.85]  2.11 [1.74, 2.54]
         VESAD tactile     2.83 [2.36, 3.29]      1.09 [0.91, 1.29]  0.69 [0.51, 0.89]
         VESAD            8.72 [7.16, 10.37]      0.47 [0.19, 0.86]  0.62 [0.16, 1.19]

                      Mouvements tête-téléphone
         Technique
         Téléphone             3.18 [2.34, 4.07]
         VESAD tactile         1.57 [1.25, 1.92]
         VESAD                 6.12 [4.86, 7.61]
```

```
In [71]: g = sns.factorplot(x=technique['label'], y=labels['distance'],
                            col=labels['category'], data=trial_distances,
                            palette=technique['palette'], kind='bar', col_wrap=2)

         g.set_titles('{col_name}') # Replace subplot titles
         for ax in g.axes:
             ax.title.set_position([0.5, -0.12])

         g.set_axis_labels('') # Custom legend
         g.set_xticklabels([])
         legend_handles = [patches.Patch(color=color, label=value)\
                           for value, color in zip(technique['categorical'],
                                                   technique['palette'])]
```

```
legend = plt.legend(handles=legend_handles, loc='center left',
                    bbox_to_anchor=(1, 1.1), title=technique['label'])
fix_legend_fontsize(legend)

g.savefig('navigation_distance.png')
```