In [1]:

```python
import math
import pandas
import operator
import plotly.express as px
from functools import reduce
from itertools import combinations, starmap

def data():
   return pandas.read_csv("data.csv", sep=";", skipinitialspace=True, quotechar='"').set_index("Specie")

print(data().sum())
data().style
```

```
Floodplain Oak    203
Tourist Bases      52
Park               63
dtype: int64
```

Out[1]:

Specie	Floodplain Oak	Tourist Bases	Park
Columba livia Gmelin, 1789	0	0	3
Streptopelia decaocto (Frivaldszky, 1838)	0	0	1
Apus apus (L.1758)	0	0	9
Alcedo atthis (L.1758)	2	0	0
Merops apiaster L.1758	6	0	0
Upupa epops L.1758	1	0	0
Jynx torquilla L.1758	1	0	0
Dendrocopos major (L.1758)	5	1	0
Dendrocopos syriacus (Hemprich et Ehrenberg, 1833)	0	0	1
Leiopicus medius (L.1758)	2	1	0
Dryobates minor	2	1	0
Riparia riparia (L.1758)	6	0	0
Hirundo rustica L.1758	0	12	7
Delichon urbicum (L.1758)	0	6	4
Motacilla alba L.1758	3	0	1
Lanius collurio L.1758	1	1	0
Sturnus vulgaris L.1758	0	3	1
Garrulus glandarius (L.1758)	1	0	0
Corvus cornix L.1758	0	0	1
Corvus corax L.1758	1	0	0
Troglodytes troglodytes (L.1758)	2	0	0
Sylvia communis Latham, 1787	3	0	0
Phylloscopus collybita (Vieillot, 1817)	8	0	0
Phylloscopus sibilatrix (Bechstein, 1793)	3	0	0
Ficedula hypoleuca (Pallas, 1764)	7	1	0
Ficedula albicollis (Temminck, 1815)	11	1	1
Phoenicurus ochruros (S.G. Gmelin, 1774)	0	0	2
Erithacus rubecula (L.1758)	4	0	0
Luscinia luscinia (L.1758)	2	0	0
Turdus merula L.1758	12	0	1
Turdus philomelos C.L. Brehm, 1831	2	1	0
Aegithalos caudatus (L.1758)	3	0	1
Periparus ater L.1758	4	1	0
Cyanistes caeruleus L.1758	3	0	0
Parus major L. 1758	21	4	2
Sitta europaea L.1758	13	3	1
Certhia familiaris L.1758	10	1	0
Passer domesticus (L.1758)	0	6	8
Passer montanus (L.1758)	16	4	12
Fringilla coelebs L.1758	20	1	1
Serinus serinus (Pallas, 1811)	0	0	1
Chloris chloris (L. 1758)	1	0	2
Carduelis carduelis (L.1758)	19	3	2
Coccothraustes coccothraustes (L.1758)	8	1	1

α-diversity (raw)

In [2]:

```python
def filter_none(vector):
    return tuple(filter(None, vector))

def no_zeros(fn):
    return lambda vector: fn(filter_none(vector))

@no_zeros
def shannon_index(vector):
    N = sum(vector)
    p_vector = map(lambda n: n / N, vector)
    entropy_vector = map(lambda p: p * math.log(p), p_vector)
    return -sum(entropy_vector)

@no_zeros
def pielou_index(vector):
    return shannon_index(vector) / math.log(len(vector))

@no_zeros
def simpson_index(vector):
    N = sum(vector)
    Ni = N * (N - 1)
    ni_vector = map(lambda n: n * (n - 1), vector)
    return sum(ni_vector) / Ni

@no_zeros
def berger_index(vector):
    return max(vector) / sum(vector)

def alpha_indices(df):
    indices = {
        "Shannon index": shannon_index,
        "Pielou index": pielou_index,
        "Simpson index": simpson_index,
        "Berger-Parker index": berger_index,
    }
    return df.apply(lambda column: tuple(map(lambda fn: fn(column), indices.values()))).set_index(pandas.Series(indices.keys()))

alpha_indices(data()).style
```

Out[2]:

	Floodplain Oak	Tourist Bases	Park
Shannon index	3.101140	2.560900	2.647309
Pielou index	0.886925	0.869741	0.856445
Simpson index	0.052919	0.088235	0.083973
Berger-Parker index	0.103448	0.230769	0.190476

Density per km^2

In [3]:

```python
def areas():
    TRANSECTS_LENGTH = pandas.Series((3000, 1000, 700), index=("Floodplain Oak", "Tourist Bases", "Park")) # m
    TRANSECTS_WIDTH = 100 # m
    return TRANSECTS_LENGTH.mul(TRANSECTS_WIDTH).div(10**6) # km^2

def density():
    return data().div(areas()) # individuals / km^2

print(density().sum())
density().style
```

```
Floodplain Oak    676.666667
Tourist Bases     520.000000
Park              900.000000
dtype: float64
```

Out[3]:

Specie	Floodplain Oak	Tourist Bases	Park
Columba livia Gmelin, 1789	0.000000	0.000000	42.857143
Streptopelia decaocto (Frivaldszky, 1838)	0.000000	0.000000	14.285714
Apus apus (L.1758)	0.000000	0.000000	128.571429
Alcedo atthis (L.1758)	6.666667	0.000000	0.000000
Merops apiaster L.1758	20.000000	0.000000	0.000000
Upupa epops L.1758	3.333333	0.000000	0.000000
Jynx torquilla L.1758	3.333333	0.000000	0.000000
Dendrocopos major (L.1758)	16.666667	10.000000	0.000000
Dendrocopos syriacus (Hemprich et Ehrenberg, 1833)	0.000000	0.000000	14.285714
Leiopicus medius (L.1758)	6.666667	10.000000	0.000000
Dryobates minor	6.666667	10.000000	0.000000
Riparia riparia (L.1758)	20.000000	0.000000	0.000000
Hirundo rustica L.1758	0.000000	120.000000	100.000000
Delichon urbicum (L.1758)	0.000000	60.000000	57.142857
Motacilla alba L.1758	10.000000	0.000000	14.285714
Lanius collurio L.1758	3.333333	10.000000	0.000000
Sturnus vulgaris L.1758	0.000000	30.000000	14.285714
Garrulus glandarius (L.1758)	3.333333	0.000000	0.000000
Corvus cornix L.1758	0.000000	0.000000	14.285714
Corvus corax L.1758	3.333333	0.000000	0.000000
Troglodytes troglodytes (L.1758)	6.666667	0.000000	0.000000
Sylvia communis Latham, 1787	10.000000	0.000000	0.000000
Phylloscopus collybita (Vieillot, 1817)	26.666667	0.000000	0.000000
Phylloscopus sibilatrix (Bechstein, 1793)	10.000000	0.000000	0.000000
Ficedula hypoleuca (Pallas, 1764)	23.333333	10.000000	0.000000
Ficedula albicollis (Temminck, 1815)	36.666667	10.000000	14.285714
Phoenicurus ochruros (S.G. Gmelin, 1774)	0.000000	0.000000	28.571429
Erithacus rubecula (L.1758)	13.333333	0.000000	0.000000
Luscinia luscinia (L.1758)	6.666667	0.000000	0.000000
Turdus merula L.1758	40.000000	0.000000	14.285714
Turdus philomelos C.L. Brehm, 1831	6.666667	10.000000	0.000000
Aegithalos caudatus (L.1758)	10.000000	0.000000	14.285714
Periparus ater L.1758	13.333333	10.000000	0.000000
Cyanistes caeruleus L.1758	10.000000	0.000000	0.000000
Parus major L. 1758	70.000000	40.000000	28.571429
Sitta europaea L.1758	43.333333	30.000000	14.285714
Certhia familiaris L.1758	33.333333	10.000000	0.000000
Passer domesticus (L.1758)	0.000000	60.000000	114.285714
Passer montanus (L.1758)	53.333333	40.000000	171.428571
Fringilla coelebs L.1758	66.666667	10.000000	14.285714
Serinus serinus (Pallas, 1811)	0.000000	0.000000	14.285714
Chloris chloris (L. 1758)	3.333333	0.000000	28.571429
Carduelis carduelis (L.1758)	63.333333	30.000000	28.571429
Coccothraustes coccothraustes (L.1758)	26.666667	10.000000	14.285714

α-diversity (by density)

In [4]:

```python
alpha_indices(density()).style
```

Out[4]:

	Floodplain Oak	Tourist Bases	Park
Shannon index	3.101140	2.560900	2.647309
Pielou index	0.886925	0.869741	0.856445
Simpson index	0.056190	0.104046	0.097511
Berger-Parker index	0.103448	0.230769	0.190476

Weighting

Estimated sampling fraction

Sampling fraction $f = n/N$, where n is the sample size, N is the community size in related stratum.

We can approximate $N = AD$, where A is the total stratum area in km^2 and $D = n/a$ is the relative density in $individuals/km^2$, a is the observed area in km^2.

Hence $f = n/AD = \frac{n}{nA/a} = \frac{na}{nA} = a/A$

In [5]:

```python
STRATUM_AREAS = pandas.Series((2.17, 0.9, 0.75), index=("Floodplain Oak", "Tourist Bases", "Park")) # km^2

def sampling_fractions():
    return areas().div(STRATUM_AREAS)

print(sampling_fractions())
print(f"Mean: {sum(sampling_fractions()) / len(sampling_fractions())}")
print(f"Global: {sum(areas()) / sum(STRATUM_AREAS)}")
```

```
Floodplain Oak    0.138249
Tourist Bases     0.111111
Park              0.093333
dtype: float64
Mean: 0.11423109745690391
Global: 0.1230366492146597
```

Weights

$W_h = f/f_h$ is the weihgt for h-th stratum, where f - global sampling fraction and f_h - sampling fraction for the stratum.

But from [Leslie Kish, "Weighting for Unequal P_i"]:

> f_h may be simple integral multipliers like $2f$ or $10f$.

For our case

> f-based weights may be less extreme and estimates for means may be weighted.

In [6]:

```python
def density_weights():
    total_densities = density().sum()
    mean_density = total_densities.sum() / len(total_densities)
    return total_densities.rdiv(mean_density)

def fraction_weights():
    norm = 1 / sampling_fractions().sum()
    normalized_fractions = sampling_fractions().mul(norm)
    mean_fraction = 1 / len(normalized_fractions)
    return normalized_fractions.rdiv(mean_fraction)

def size_weights():
    totals = data().sum()
    mean = sum(totals) / len(totals)
    return totals.rdiv(mean)

# Most reliable weight combination
def weights():
    products = density_weights().mul(fraction_weights())
    mean = sum(products) / len(products)
    return products.rdiv(mean)

print("Density weights")
print(density_weights())
print()
print("Fraction weights")
print(fraction_weights())
print()
print("Sample size (N) weights")
print(size_weights())
print()
print("Most reliable weight combination")
print(weights())
```

```
Density weights
Floodplain Oak    1.032841
Tourist Bases     1.344017
Park              0.776543
dtype: float64

Fraction weights
Floodplain Oak    0.826272
Tourist Bases     1.028080
Park              1.223905
dtype: float64

Sample size (N) weights
Floodplain Oak    0.522167
Tourist Bases     2.038462
Park              1.682540
dtype: float64

Most reliable weight combination
Floodplain Oak    1.244259
Tourist Bases     0.768485
Park              1.117259
dtype: float64
```

α-diversity (weighted)

$H = -W_h \sum p_i lnp_i$

This is not true now:

> $W_h = 1/f_h = N_h/n_h$ is h-th stratum weight
>
> $\sum W_h = 1$

In [7]:

```python
alpha_indices(data()).mul(weights()).style
```

Out[7]:

	Floodplain Oak	Tourist Bases	Park
Shannon index	3.858623	1.968014	2.957730
Pielou index	1.103565	0.668383	0.956871
Simpson index	0.065845	0.067808	0.093820
Berger-Parker index	0.128716	0.177343	0.212811

α-diversity (new entropy estimator)

[Entropy and the species accumulation curve, Chao A.]

(4b) - just to explain the code block below

$H = \sum_{k=1}^{n-1} \frac{1}{k} \Delta(k) + \dots$

(7)

$H = \sum_{1 \le X_i \le n-1} \frac{X_i}{n} (\sum_{k=X_i}^{n-1} \frac{1}{k}) + \frac{f_1}{n} (1-A)^{1-n} * (-log(A) - \sum_{r=1}^{n-1} \frac{1}{r} (1-A)^r)$

where:

n - sample size

X_i - i-th specie frequency in the sample $(X_1, X_2, \dots, X_n)$, when $\sum_{X_i > 0} = n$. Denoted as `x` in the code.

f_1 - number of singletons in the sample

f_2 - number of doubletons in the sample

$A = 1$ if $f_1 = f_2 = 0$

$A = 2f_2/((n-1) * f_1 + 2f_2)$ if $f_2 > 0$

$A = 2/((n-1) * (f_1 - 1) + 2)$ if $f_1 > 0$ and $f_2 = 0$

In [8]:

```python
def mean_relative_frequency(n, f1, f2):
    if 0 == f1 == f2:
        return 1
    if 0 < f2:
        return 2*f2 / ((n-1)*f1 + 2*f2)
    return 2 / ((n-1)*(f1-1) + 2)

def shannon_entropy(vector):
    n = sum(vector)

    # First part of H for k < n, see (4b)
    x_vector = [x for x in vector if 1 <= x <= n-1]
    H_less_n = sum((x/n) * sum(1/k for k in range(x, n)) for x in x_vector)

    f1 = tuple(vector).count(1) # singletons
    f2 = tuple(vector).count(2) # doubletons
    A = mean_relative_frequency(n, f1, f2)
    H_tail = (f1/n) * pow((1-A), 1-n) * (-math.log(A) - sum(pow(1-A, r) / r for r in range(1, n)))
    return H_less_n + H_tail

print(data().apply(shannon_entropy))
```

```
Floodplain Oak    3.195581
Tourist Bases     3.067454
Park              2.979396
dtype: float64
```

In [9]:

```python
def pielou_evenness(vector):
    return shannon_entropy(vector) / math.log(len(vector))

def entropy_weight(vector):
    return shannon_entropy(vector) / shannon_index(vector)

def berger_dominance(vector):
    return entropy_weight(vector) * max(vector) / sum(vector)

def alpha_indices_new(df):
    indices = {
        "Shannon entropy": shannon_entropy,
        "Pielou evenness": pielou_evenness,
        "Berger-Parker index": berger_dominance,
    }
    return df.apply(lambda column: tuple(map(lambda fn: fn(column), indices.values()))).set_index(pandas.Series(indices.keys()))

alpha_indices_new(data()).style
```

Out[9]:

	Floodplain Oak	Tourist Bases	Park
Shannon entropy	3.195581	3.067454	2.979396
Pielou evenness	0.844456	0.810597	0.787327
Berger-Parker index	0.106599	0.276416	0.214370

β-diversity

[Chao, Abundance-Based Similarity Indices]

(1)

$U = \sum_{i=1}^{D_{12}} \frac{X_i}{n} + \frac{m-1}{m} \frac{f_{+1}}{2f_{+2}} \sum_{i=1}^{D_{12}} \frac{X_i}{n} I(Y_i = 1)$

(2)

$V = \sum_{i=1}^{D_{12}} \frac{Y_i}{m} + \frac{n-1}{n} \frac{f_{1+}}{2f_{2+}} \sum_{i=1}^{D_{12}} \frac{Y_i}{m} I(X_i = 1)$

U and V - total relative abundances for sample 1 $(X_1, X_2, \dots, X_{S_1})$ and sample 2 $(Y_1, Y_2, \dots, Y_{S_2})$, where $\sum X_i = n$ and $\sum Y_i = m$.

Assume that D_1 and D_2 species observed in sample 1 and 2 respectively and D_{12} are shared species.

$f_{1+} = \sum_{i=1}^{D_{12}} I(X_i = 1, Y_i \ge 1)$ - singletons in sample 1, f_{+1} and others - by analogue.

When f_{+2} or f_{2+} is 0, then 1 is the replacement.

If U or V is greater than 1 then it is replaced by 1.

Jaccard index:

$C_j = \frac{UV}{U+V-UV}$

Sorensen index:

$C_s = \frac{2UV}{U+V}$

In [10]:

```python
def sample_pair(fn):
    # TODO: remove
    # Gets a sample pair `s1` and `s2` from the `data` frame by column names `c1` and `c2` and return `fn(s1, s2)`.
    return lambda c1, c2: fn(data()[c1], data()[c2])

def uv_pair(fn):
    def decorator(s1, s2):
        # Calculates `u` for sample `s1` and `v` for sample `s2` and returns `fn(u, v)`,
        # where `u` and `v` are total relative abundances.

        n = sum(s1)
        m = sum(s2)
        assert n and m

        shared_species = tuple(filter(lambda species: s1[species] and s2[species], data().index))
        shared_s1 = s1[s1.index.isin(shared_species)]
        shared_s2 = s2[s2.index.isin(shared_species)]

        assert 0 < len(shared_s1)
        assert all(shared_s1) and all(shared_s2)
        assert len(shared_s1) == len(shared_s2)
        assert len(shared_s1) <= n
        assert len(shared_s2) <= m

        unajusted_u = sum(map(lambda x: x / n, shared_s1))
        unajusted_v = sum(map(lambda y: y / m, shared_s2))

        f_1s = len(tuple(filter(lambda x: x == 1, shared_s1)))
        f_s1 = len(tuple(filter(lambda y: y == 1, shared_s2)))
        assert f_1s and f_s1

        f_2s = max(1, len(tuple(filter(lambda x: x == 2, shared_s1))))
        f_s2 = max(1, len(tuple(filter(lambda y: y == 2, shared_s2))))
        assert f_2s and f_s2

        u = min(1, unajusted_u + ((m - 1) / m) * (f_s1 / (2 * f_s2)) * sum(int(shared_s2[specie] == 1) * shared_s1[specie] / n for specie in shared_species))
        v = min(1, unajusted_v + ((n - 1) / n) * (f_1s / (2 * f_2s)) * sum(int(shared_s1[specie] == 1) * shared_s2[specie] / m for specie in shared_species))

        assert u <= 1 and v <= 1

        return fn(u, v)
    return decorator

@sample_pair
@uv_pair
def sorensen(u, v):
    return 2*u*v / (u + v)

@sample_pair
@uv_pair
def jaccard(u, v):
    return u*v / (u + v - u*v)

def beta_indices():
    column_pairs = tuple(combinations(data().columns, 2))
    return pandas.DataFrame(
        data = {
            "Sorensen index": starmap(sorensen, column_pairs),
            "Jaccard index": starmap(jaccard, column_pairs),
        },
        index = (f"{c1} / {c2}" for c1, c2 in column_pairs)
    )

beta_indices().style
```

Out[10]:

	Sorensen index	Jaccard index
Floodplain Oak / Tourist Bases	0.652253	0.483959
Floodplain Oak / Park	0.584191	0.412620
Tourist Bases / Park	0.826963	0.704976