

Accessible Algorithms

@maxhumber

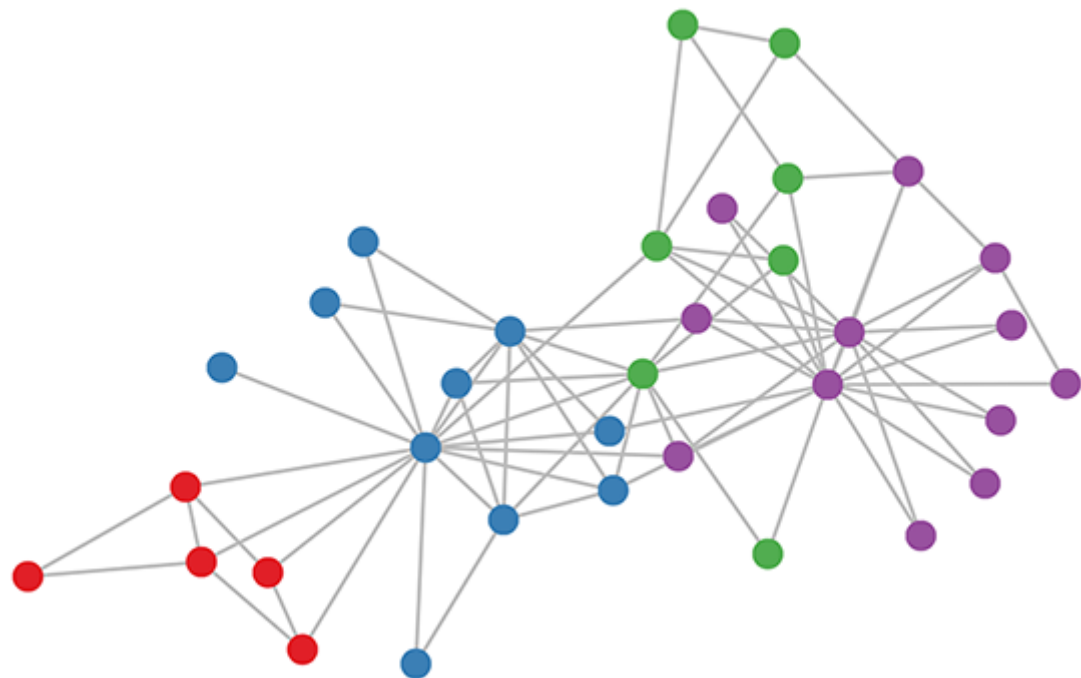
PyCon 

2018-11-10

> save the environment

> defeat your enemies







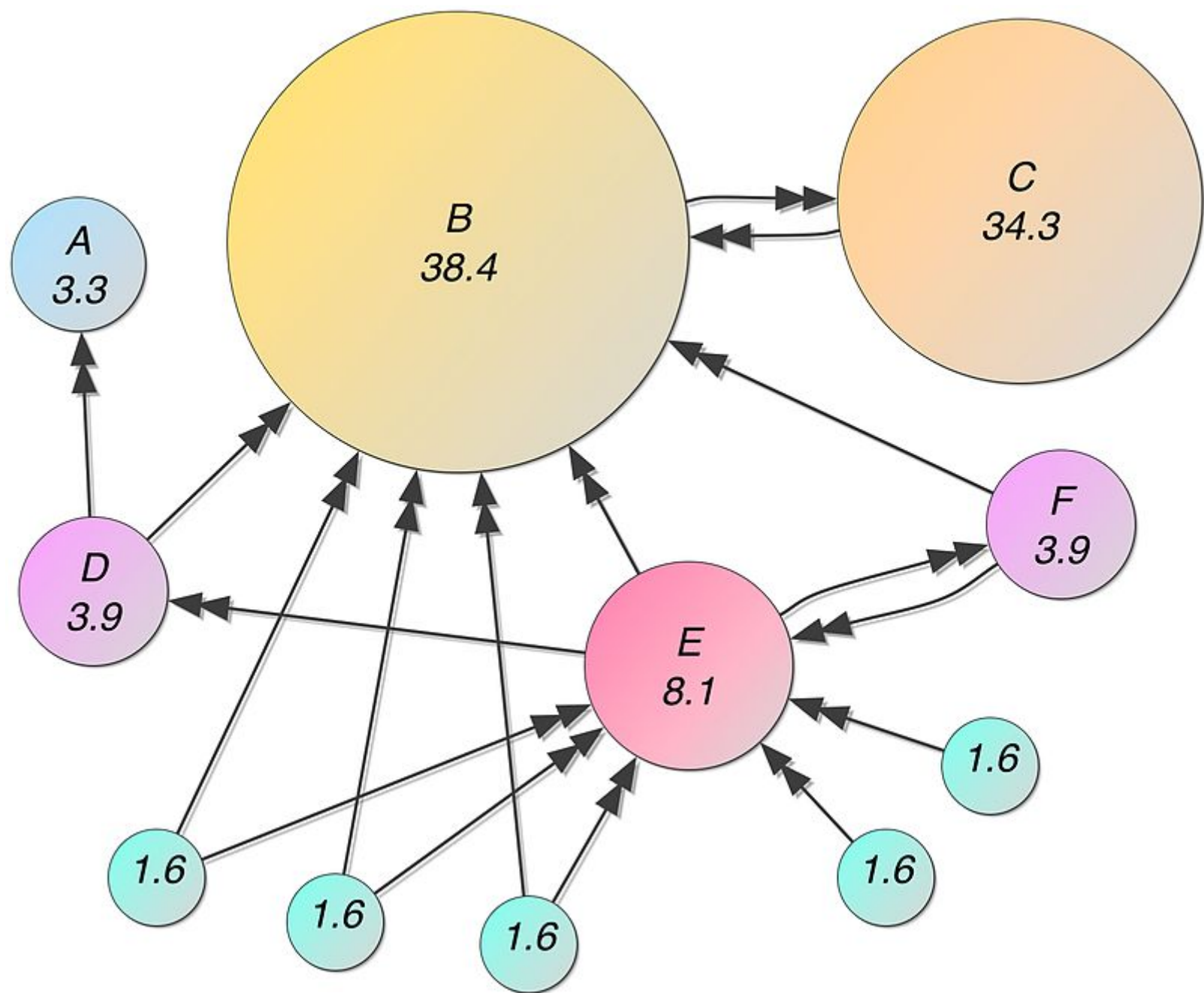
The PageRank Citation Ranking: Bringing Order to the Web

January 29, 1998

Abstract

The importance of a Web page is an inherently subjective matter, which depends on the readers interests, knowledge and attitudes. But there is still much that can be said objectively about the relative importance of Web pages. This paper describes PageRank, a method for rating Web pages objectively and mechanically, effectively measuring the human interest and attention devoted to them.

We compare PageRank to an idealized random Web surfer. We show how to efficiently compute PageRank for large numbers of pages. And, we show how to apply PageRank to search and to user navigation.





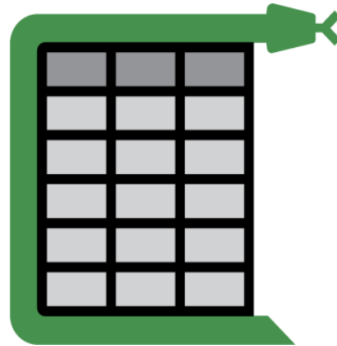
Extinction Countdown

Can Google's page-rank algorithm help save endangered species and ecosystems?

By John Platt

...a species is important if important species rely on it for their survival...

Species	Species they feed on
Shark	Sea otter
Sea otter	Sea stars, sea urchins, large crabs, large fish and octopus, abalone
Sea stars	Abalone, small herbivorous fishes, sea urchins
Sea urchins	Kelp, sessile invertebrates, drift algae and dead animals
Abalone	Drift algae and dead animals
Large crabs	Sea stars, smaller predatory fishes and invertebrates, drift algae and dead animals, small herbivorous fishes and invertebrates, kelp
Smaller predatory fishes	Sessile invertebrates, planktonic invertebrates
Small herbivorous fishes and invertebrates	Kelp
Kelp	- -
Large fish and octopus	Smaller predatory fishes and invertebrates
Sessile invertebrates	Microscopic planktonic algae, planktonic invertebrates
algae and dead animals	Kelp, sessile invertebrates
Planktonic invertebrates	Microscopic planktonic algae
Microscopic planktonic algae	- -



Camelot: PDF Table Extraction for Humans

build failing docs passing codecov 87% pypi v0.3.2 license MIT License python 2.7 | 3.5 | 3.6 | 3.7 chat on gitter

Camelot is a Python library that makes it easy for *anyone* to extract tables from PDF files!

Note: You can also check out [Excalibur](#), which is a web interface for Camelot!

```
In [1]: # !pip install camelot-py[cv]

import pandas as pd
import camelot

tables = camelot.read_pdf('data/mmnp201166p22.pdf', pages='5')
tables[0].parsing_report
```

```
Out[1]: {'accuracy': 100.0, 'whitespace': 0.0, 'order': 1, 'page': 5}
```

```
In [2]: print('>', len(tables))

df = tables[0].df
df
```

> 1

Out[2]:

	0	1
0	Species	Species they feed on
1	Shark	Sea otter
2	Sea otter	Sea stars, sea urchins, large crabs, large fish...
3	Sea stars	Abalone, small herbivorous fishes, sea urchins
4	Sea urchins	Kelp, sessile invertebrates, drift algae and d...
5	Abalone	Drift algae and dead animals
6	Large crabs	Sea stars, smaller predatory fishes and inverte...
7	Smaller predatory fishes	Sessile invertebrates, planktonic invertebrates
8	Small herbivorous fishes and invertebrates	Kelp
9	Kelp	--
10	Large fish and octopus	Smaller predatory fishes and invertebrates
11	Sessile invertebrates	Microscopic planktonic algae, planktonic inverte- b...

```
In [3]: df.columns = ['pred', 'prey']
df = df.reindex(df.index.drop(0))

mapping = {
    'ani-mals': 'animals',
    'and dead animals': '',
    'drift ': '',
    'andoctopus': 'and octopus',
    'microscopicplanktonicalgae': 'microscopic planktonic algae',
    'planktonicinverte-brates': 'planktonic invertebrates',
    'andinvertebrates': 'and invertebrates',
    'and invertebrates': '',
    'fishesand': 'fishes and',
    'fi': 'fi',
}
```

```
In [4]: import re

print(mapping['microscopicplanktonicalgae'])

def fix_text(text, mapping):
    for k, v in mapping.items():
        t = re.compile(re.escape(k), re.IGNORECASE)
        text = t.sub(v, text)
    return text

df.pred = df.pred.apply(lambda x: fix_text(x.lower(), mapping))
df.prey = df.prey.apply(lambda x: fix_text(x.lower(), mapping))
```

microscopic planktonic algae

In [7]: `df.head()`

Out[7]:

	pred	prey
1	shark	sea otter
2	sea otter	sea stars, sea urchins, large crabs, large fis...
3	sea stars	abalone, small herbivorous fishes, sea urchins
4	sea urchins	kelp, sessile invertebrates, algae
5	abalone	algae


```
In [8]: (
    df.prey
    .str
    .split(',', expand=True)
    .stack()
    .reset_index(drop=True, level=1)
    .rename('prey')
).head()
```

```
Out[8]: 1          sea otter
        2          sea stars
        2      sea urchins
        2      large crabs
        2  large fish and octopus
        Name: prey, dtype: object
```

```
In [9]: df = df.drop('prey', axis=1).join(
        df.prey
        .str
        .split(',', expand=True)
        .stack()
        .reset_index(drop=True, level=1)
        .rename('prey')
    ).reset_index(drop=True)

df.head()
```

Out[9]:

	pred	prey
0	shark	sea otter
1	sea otter	sea stars
2	sea otter	sea urchins
3	sea otter	large crabs
4	sea otter	large fish and octopus

```
In [10]: df = df[df['prey'] != '--']  
df.loc[:, 'prey'] = df['prey'].str.strip()  
df.loc[:, 'pred'] = df['pred'].str.strip()
```

```
In [12]: df.to_csv('data/food_web.csv', index=False)

df.head()
```

Out[12]:

	pred	prey
0	shark	sea otter
1	sea otter	sea stars
2	sea otter	sea urchins
3	sea otter	large crabs
4	sea otter	large fish and octopus

```
In [13]: import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
%matplotlib inline
```

```
In [14]: G = nx.DiGraph()

G.add_edge('shark', 'sea otter')
G.add_edge('sea otter', 'sea stars')

G.nodes()
G.edges()
```

```
Out[14]: OutEdgeView([('shark', 'sea otter'), ('sea otter', 'sea stars')])
```

```
In [15]: df = pd.read_csv('data/food_web.csv')
```

```
G = nx.from_pandas_edgelist(  
    df,  
    source='pred',  
    target='prey',  
    create_using=nx.DiGraph  
)
```

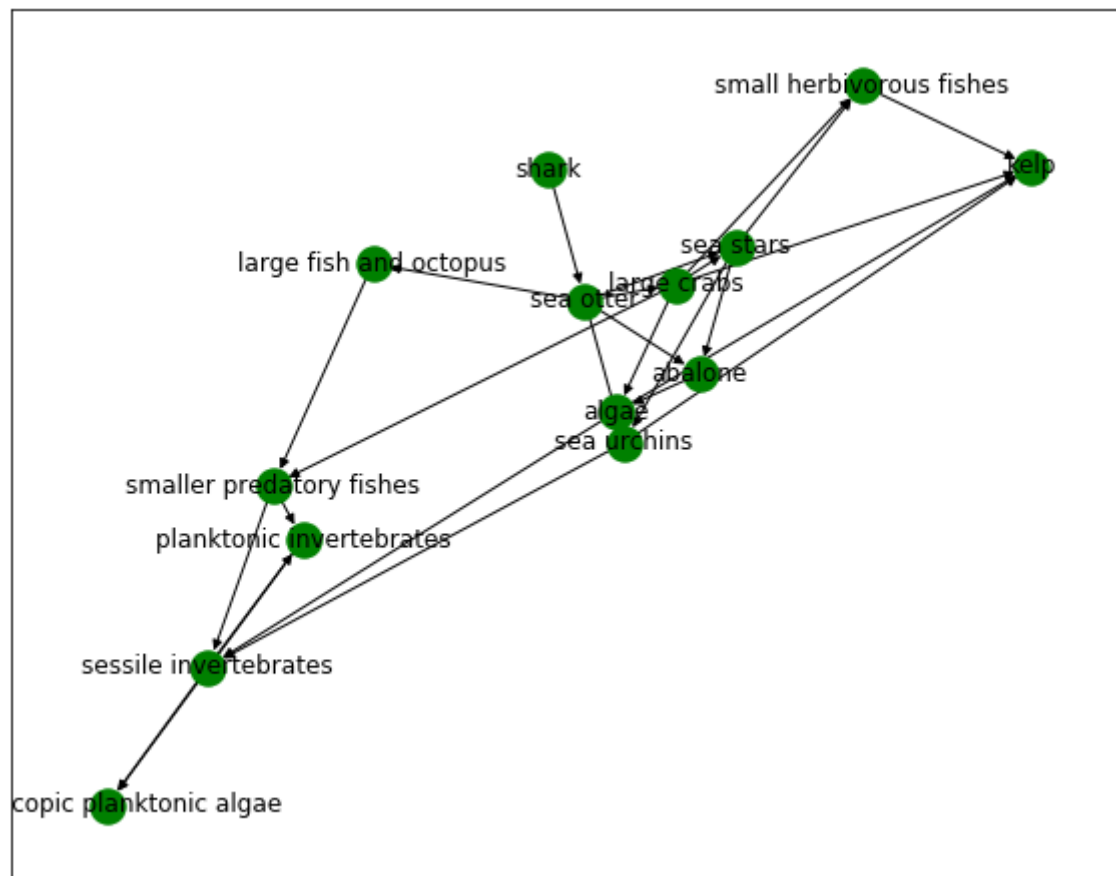
```
In [16]: from pprint import pprint  
        pprint(list(G.nodes))  
  
['shark',  
 'sea otter',  
 'sea stars',  
 'sea urchins',  
 'large crabs',  
 'large fish and octopus',  
 'abalone',  
 'small herbivorous fishes',  
 'kelp',  
 'sessile invertebrates',  
 'algae',  
 'smaller predatory fishes',  
 'planktonic invertebrates',  
 'microscopic planktonic algae']
```

```
In [17]: pprint(list(G.edges))
```

```
[('shark', 'sea otter'),
 ('sea otter', 'sea stars'),
 ('sea otter', 'sea urchins'),
 ('sea otter', 'large crabs'),
 ('sea otter', 'large fish and octopus'),
 ('sea otter', 'abalone'),
 ('sea stars', 'abalone'),
 ('sea stars', 'small herbivorous fishes'),
 ('sea stars', 'sea urchins'),
 ('sea urchins', 'kelp'),
 ('sea urchins', 'sessile invertebrates'),
 ('sea urchins', 'algae'),
 ('large crabs', 'sea stars'),
 ('large crabs', 'smaller predatory fishes'),
 ('large crabs', 'algae'),
 ('large crabs', 'small herbivorous fishes'),
 ('large crabs', 'kelp'),
 ('large fish and octopus', 'smaller predatory fishes'),
 ('abalone', 'algae'),
 ('small herbivorous fishes', 'kelp'),
 ('sessile invertebrates', 'microscopic planktonic algae'),
 ('sessile invertebrates', 'planktonic invertebrates'),
 ('algae', 'kelp'),
 ('algae', 'sessile invertebrates'),
 ('smaller predatory fishes', 'sessile invertebrates'),
 ('smaller predatory fishes', 'planktonic invertebrates'),
 ('planktonic invertebrates', 'microscopic planktonic algae')]
```



```
In [18]: np.random.seed(1)
plt.figure(figsize=(10, 8))
nx.draw_networkx(G, node_color='green')
plt.xticks([])
plt.yticks([]);
```



$$PR(A) = (1 - d) + d\left(\frac{PR(T_1)}{C(T_1)} + \dots + \frac{PR(T_n)}{C(T_n)}\right)$$

where

- $PR(A)$ is the PageRank of page A
- $PR(T_i)$ is the PageRank of pages T_i which link to page A
- $C(T_i)$ is the number of outbound links on page T_i and
- d is a damping factor which can be set between 0 and 1

```

In [19]: def pagerank(G, alpha=0.85, max_iter=100, tol=1.0e-6):

    W = nx.stochastic_graph(G)
    N = len(W)
    x = {n: 1/N for n in W.nodes}

    p = x
    dangling_weights = p
    dangling_nodes = [n for n in W if W.out_degree(n) == 0.0]

    for _ in range(max_iter):
        xlast = x
        x = {key: 0 for key in x}
        danglesum = alpha * sum([xlast[n] for n in dangling_nodes])
        for n in x:
            for nbr in W[n]:
                x[nbr] += alpha * xlast[n] * W[n][nbr]['weight']
            x[n] += danglesum * dangling_weights.get(n, 0) + (1.0 - alpha) * p.get
(n, 0)
        err = sum([abs(x[n] - xlast[n]) for n in x])
        if err < N * tol:
            return x

```

```
In [20]: pprint(pagerank(G, alpha=0.85))
```

```
{'abalone': 0.04940106801052845,  
'algae': 0.09052025512398879,  
'kelp': 0.1268076024301348,  
'large crabs': 0.03710188718650615,  
'large fish and octopus': 0.03710188718650615,  
'microscopic planktonic algae': 0.16160847090331737,  
'planktonic invertebrates': 0.10253360633998779,  
'sea otter': 0.052216411290389245,  
'sea stars': 0.0434091571287912,  
'sea urchins': 0.04940106801052845,  
'sessile invertebrates': 0.1087727093009147,  
'shark': 0.028225268889463057,  
'small herbivorous fishes': 0.046831719655770404,  
'smaller predatory fishes': 0.06606888854317333}
```

```
In [21]: pprint(nx.pagerank(G, alpha=0.85))
```

```
{'abalone': 0.04940106801052845,  
'algae': 0.09052025512398879,  
'kelp': 0.1268076024301348,  
'large crabs': 0.03710188718650615,  
'large fish and octopus': 0.03710188718650615,  
'microscopic planktonic algae': 0.16160847090331737,  
'planktonic invertebrates': 0.10253360633998779,  
'sea otter': 0.052216411290389245,  
'sea stars': 0.0434091571287912,  
'sea urchins': 0.04940106801052845,  
'sessile invertebrates': 0.1087727093009147,  
'shark': 0.028225268889463057,  
'small herbivorous fishes': 0.046831719655770404,  
'smaller predatory fishes': 0.06606888854317333}
```

```
In [22]: pageranks = pagerank(G, alpha=0.85)
         for g in G.nodes():
             G.nodes[g]['name'] = g
             G.nodes[g]['pagerank'] = round(pageranks[g], 4)
```

```
In [23]: G.nodes['shark']
```

```
Out[23]: {'name': 'shark', 'pagerank': 0.0282}
```

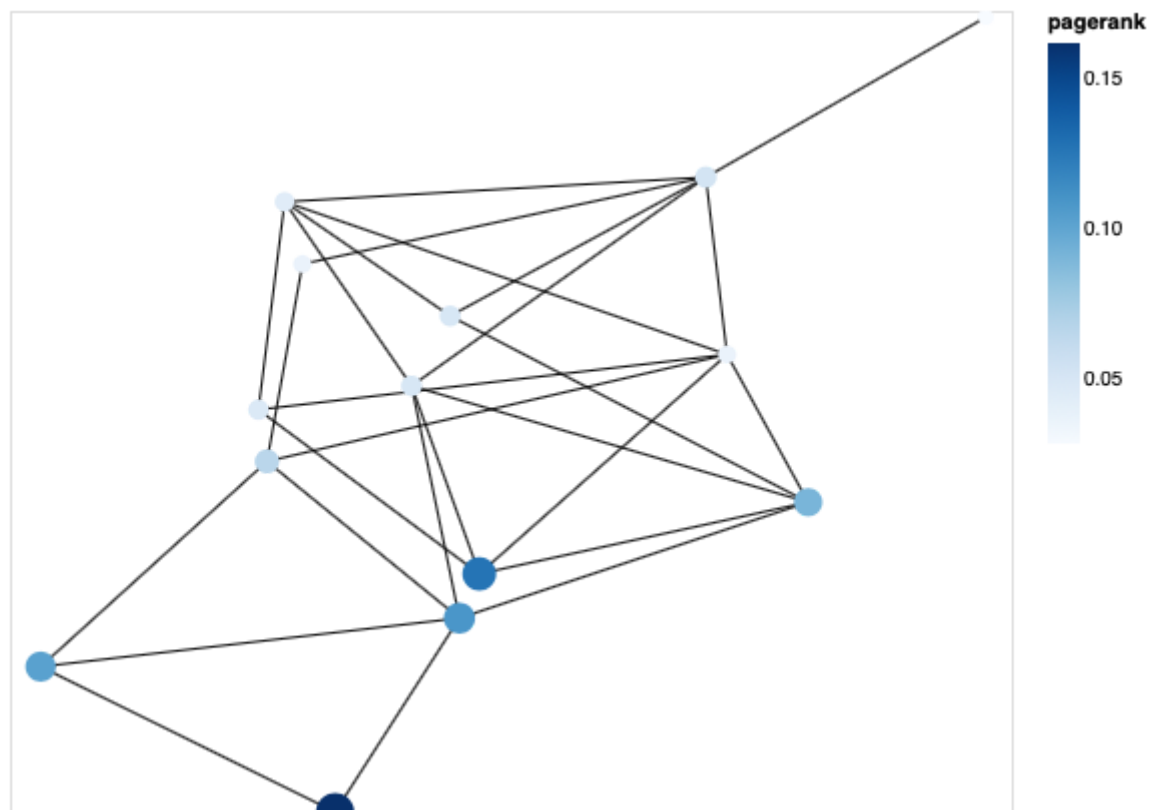
```
In [24]: import altair as alt
import nx_altair as nxa
alt.renderers.enable('notebook')

pos = nx.kamada_kawai_layout(G)

pr_viz = nxa.draw_networkx(
    G,
    pos=pos,
    node_tooltip=['name', 'pagerank'],
    node_color='pagerank',
    node_size='pagerank',
    cmap='blues'
)
```

```
In [25]: pr_viz.interactive().properties(width=500, height=400)
```

Out[25]:



"This approach contrasts with other ways of looking at ecosystems, which use a 'hub' approach to rank species based on the number of other species that are directly linked to it through the food web ... The 'PageRank' way of looking at ecosystems makes the species that goes extinct first the most important because it would result in further extinctions down the line."


```
In [26]: in_degree = dict(G.in_degree)
        pprint(in_degree)
```

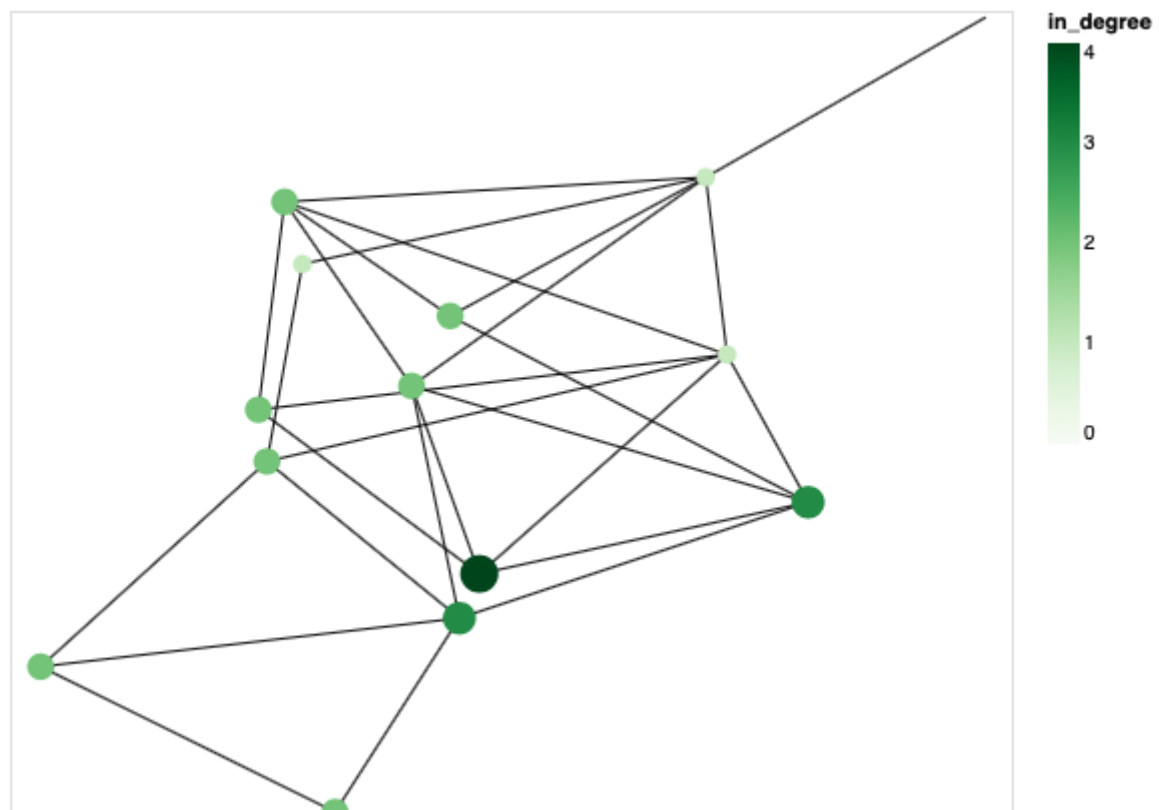
```
{'abalone': 2,
 'algae': 3,
 'kelp': 4,
 'large crabs': 1,
 'large fish and octopus': 1,
 'microscopic planktonic algae': 2,
 'planktonic invertebrates': 2,
 'sea otter': 1,
 'sea stars': 2,
 'sea urchins': 2,
 'sessile invertebrates': 3,
 'shark': 0,
 'small herbivorous fishes': 2,
 'smaller predatory fishes': 2}
```

```
In [27]: for g in G.nodes():
          G.nodes[g]['name'] = g
          G.nodes[g]['in_degree'] = in_degree[g]

hub_viz = nxa.draw_networkx(
    G,
    pos=pos,
    node_tooltip=['name', 'in_degree'],
    node_color='in_degree',
    node_size='in_degree',
    cmap='greens'
)
```

```
In [28]: hub_viz.interactive().properties(width=500, height=400)
```

Out[28]:



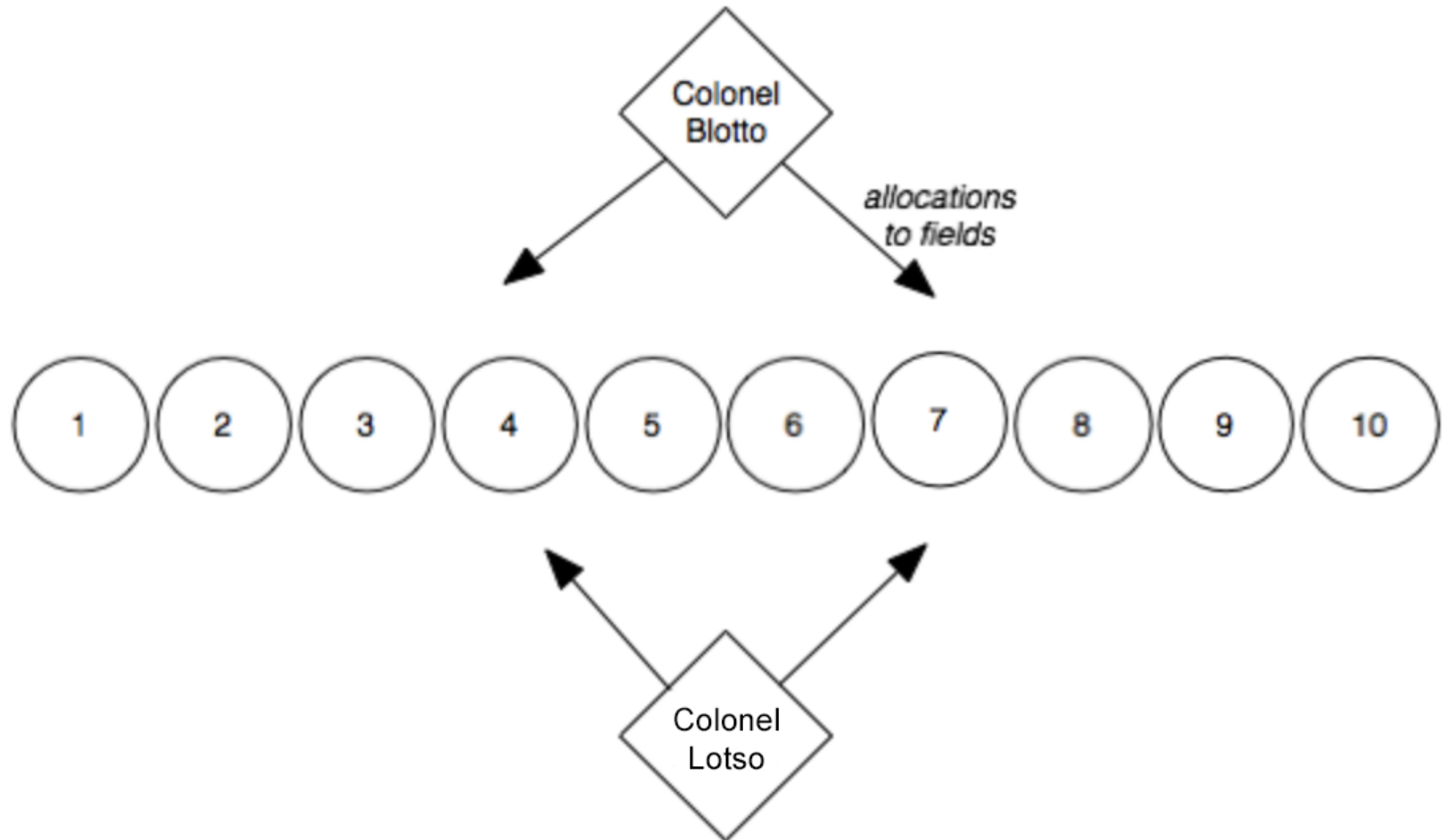


KEEP
CALM
AND
SAVE THE
WHALES

microscopic planktonic algae











































adding battlefields increases the number of interactions (dimensions) and improves the chances of an upset.

Roster Positions:	C, C, LW, LW, RW, RW, D, D, D, D, G, G, BN, BN, BN, BN, IR, IR+
Send unjoined players email reminders:	Yes
Forwards/Defenseemen Stat Categories:	Goals (G), Assists (A), Plus/Minus (+/-), Powerplay Points (PPP), Shots on Goal (SOG), Hits (HIT), Blocks (BLK)
Goaltenders Stat Categories:	Wins (W), Goals Against Average (GAA), Saves (SV), Save Percentage (SV%), Shutouts (SHO)

G	A	+/-	PPP	SOG	HIT	BLK	W	GA*	GAA	SV	SA*	SV%	SHO	Score
-	-	-	-	-	-	-	-	-	-	-	-	-	-	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

*Non-scoring stats

Name				Avg Pick ^	Avg Round	Percent Drafted
☆	Connor McDavid	Edm - C	 	1.1	1.0	100%
☆	Alex Ovechkin	Was - LW 4:00 pm vs Cls	 	2.7	1.0	100%
☆	Nikita Kucherov	TB - RW	 	3.0	1.0	100%
☆	Sidney Crosby	Pit - C	 	4.0	1.0	100%
☆	Brad Marchand	Bos - LW	 	6.3	1.0	100%
☆	Patrik Laine	Wpg - RW 5:00 pm vs Col	 	7.6	1.0	100%
☆	Nathan MacKinnon	Col - C 5:00 pm @ Wpg	 	8.7	1.1	100%
☆	Patrick Kane	Chi - RW	 	9.3	1.1	100%
☆	John Tavares	Tor - C 4:00 pm vs NJ	 	9.4	1.1	100%
☆	Auston Matthews	Tor - C 4:00 pm vs NJ	 	9.7	1.2	100%
☆	Evgeni Malkin	Pit - C	 	10.5	1.4	100%
☆	Tyler Seguin	Dal - C	 	12.2	1.4	100%
☆	Andrei Vasilevskiy	TB - G	 	13.1	1.7	100%
☆	David Pastrnak	Bos - RW	 	15.2	1.9	100%
☆	Jamie Benn	Dal - C,LW	 	16.3	2.0	100%
☆	Taylor Hall	NJ - LW 4:00 pm @ Tor	 	16.4	1.9	100%
☆	Steven Stamkos	TB - C,RW	 	17.8	2.0	100%
☆	Brent Burns	SJ - D 5:00 pm @ StL	 	18.9	2.1	100%



```
In [29]: import pandas as pd

CATEGORIES = [
    'goals',
    'assists',
    'plus_minus',
    'powerplay_points',
    'shots_on_goal',
    'hits',
    'blocks',
    'wins',
    'goals_against_average',
    'saves',
    'save_percentage',
    'shutouts'
]

raw = pd.read_csv('data/nhl_draft_2018.csv')
df = raw.copy()
```

```
In [30]: import numpy as np
```

```
np.random.seed(1)  
df.sample(10)
```

```
Out[30]:
```

	name	position	adp	goals	assists	plus_minus	power
114	T.J. Oshie	RW	120.0	22.3825	32.615000	15.1700	15.2500
85	Tyson Barrie	D	88.0	14.3450	44.637500	-9.8625	23.2500
97	Morgan Rielly	D	102.0	7.6025	45.715000	6.8000	20.7500
160	Dustin Brown	RW	168.0	23.8975	28.967500	5.8775	8.2500
35	Frederik Andersen	G	36.0	0.0000	0.000000	0.0000	0.0000
54	Jonathan Marchessault	C	55.0	30.5600	42.833333	19.2800	12.3333
124	Sean Couturier	C	131.0	21.1325	30.895000	11.4575	9.7500
19	Brent Burns	D	20.0	18.0700	50.690000	0.8725	24.2500
108	Reilly Smith	LW	114.0	24.6100	38.962500	22.7675	11.2500
125	Antti Raanta	G	132.0	0.0000	0.000000	0.0000	0.0000

```
In [31]: from sklearn.model_selection import train_test_split

target = 'adp'

y = df[target].values
X = df.drop(target, axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```
In [32]: from sklearn.preprocessing import LabelBinarizer, StandardScaler
from sklearn_pandas import DataFrameMapper
```

```
mapper = DataFrameMapper([
    ('position', LabelBinarizer()),
    (['goals'], StandardScaler()),
    (['assists'], StandardScaler()),
    (['plus_minus'], StandardScaler()),
    (['powerplay_points'], StandardScaler()),
    (['shots_on_goal'], StandardScaler()),
    (['hits'], StandardScaler()),
    (['blocks'], StandardScaler()),
    (['wins'], StandardScaler()),
    (['goals_against_average'], StandardScaler()),
    (['saves'], StandardScaler()),
    (['save_percentage'], StandardScaler()),
    (['shutouts'], StandardScaler())
], df_out=True)
```

```
X_train = mapper.fit_transform(X_train)
X_test = mapper.transform(X_test)
```

In [33]: **from sklearn.linear_model import** LinearRegression

```
lr = LinearRegression()
```

```
lr.fit(X_train, y_train)
```

```
lr.predict(X_test)[:10]
```

Out[33]: array([58.98562858, 111.85216118, 137.73149763, 122.46993907,
50.0146501 , 95.60675443, 89.65006282, -0.88417031,
175.05231979, 109.94944423])



In [34]: *#!/pip install mord*

```
import mord
```

```
model = mord.OrdinalRidge(fit_intercept=False)
```

```
model.fit(X_train, y_train)
```

```
model.predict(X_test)[:5]
```

Out[34]: array([45., 105., 139., 117., 64.])

```
In [35]: compare = pd.DataFrame({
        'true': y_test,
        'pred': model.predict(X_test)
    })

compare.head()
```

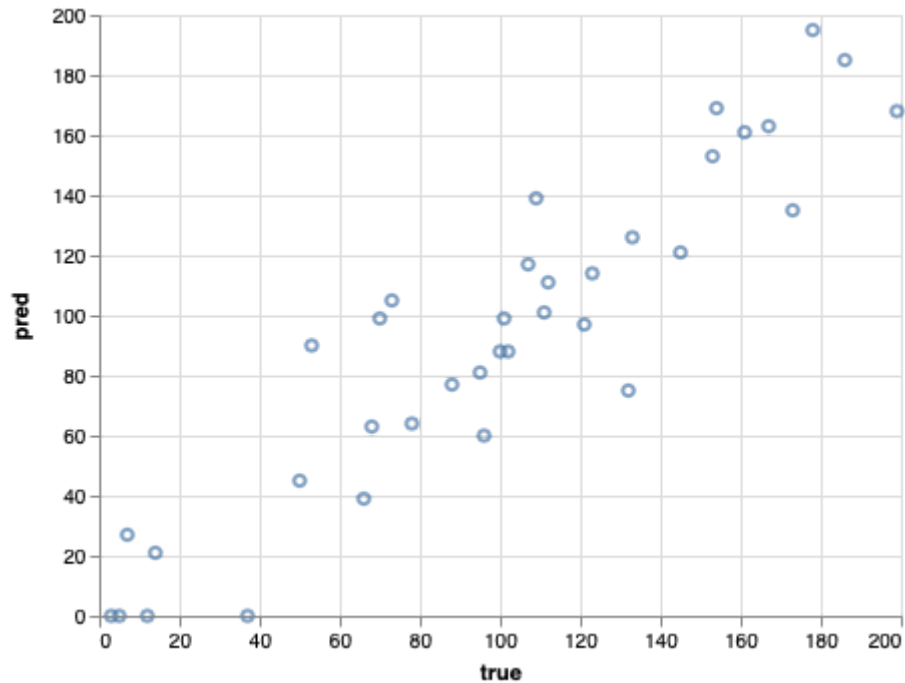
Out[35]:

	true	pred
0	50.0	45.0
1	73.0	105.0
2	109.0	139.0
3	107.0	117.0
4	78.0	64.0

```
In [36]: import altair as alt
alt.renderers.enable('notebook')

(
  alt.Chart(compare)
    .mark_point()
    .encode(
      x='true',
      y='pred'
    )
)
```

Out[36]:



```
In [38]: bias = pd.DataFrame({
          'feature': mapper.transformed_names_,
          'coef': model.coef_
        }).sort_values('coef')

bias = bias[~bias.feature.str.contains('position')]
```

```
In [39]: bias
```

```
Out[39]:
```

	feature	coef
12	wins	-44.755466
5	goals	-35.614943
6	assists	-33.406105
9	shots_on_goal	-18.803191
16	shutouts	-16.977435
8	powerplay_points	-15.200296
13	goals_against_average	-10.403468
7	plus_minus	-7.260427
14	saves	-5.824131
10	hits	-5.062121
15	save_percentage	-3.811621
11	blocks	-3.203381

Underdogs can change the odds of winning simply by changing the basis of competition.

G	A	+/-	PPP	SOG	HIT	BLK	W	GA*	GAA	SV	SA*	SV%	SHO	Score
-	-	-	-	-	-	-	-	-	-	-	-	-	-	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	0

*Non-scoring stats

```
In [40]: df.head()
```

```
Out[40]:
```

	name	position	adp	goals	assists	plus_minus	powerplay_po
0	Connor McDavid	C	1.0	40.2750	69.665000	13.217500	19.75
1	Nikita Kucherov	RW	2.0	41.5150	56.670000	17.495000	28.50
2	Alex Ovechkin	LW	3.0	50.2300	39.236667	17.126667	21.00
3	Sidney Crosby	C	4.0	35.7775	62.445000	14.142500	28.75
4	Brad Marchand	LW	5.0	38.3900	52.975000	19.537500	20.00

```
In [41]: # GAA is a bad thing, need to reverse
df['goals_against_average'] = -df['goals_against_average']

df[CATEGORIES] = (
    df
    [CATEGORIES]
    .apply(lambda x: (x - x.min()) / (x.max() - x.min()))
)
```



```
In [42]: df.head()
```

```
Out[42]:
```

	name	position	adp	goals	assists	plus_minus	powerplay_po
0	Connor McDavid	C	1.0	0.801812	1.000000	0.760291	0.470238
1	Nikita Kucherov	RW	2.0	0.826498	0.813464	0.838983	0.678571
2	Alex Ovechkin	LW	3.0	1.000000	0.563219	0.832207	0.500000
3	Sidney Crosby	C	4.0	0.712274	0.896361	0.777308	0.684524
4	Brad Marchand	LW	5.0	0.764284	0.760425	0.876558	0.476190

```
In [43]: def blotto(x, out_range=[0.80, 1]):
          domain = np.min(x), np.max(x)
          y = (x - (domain[1] + domain[0]) / 2) / (domain[1] - domain[0])
          return y * (out_range[1] - out_range[0]) + (out_range[1] + out_range[0]) / 2

bias['mod'] = bias[['coef']].apply(lambda x: blotto(x, (0.8, 1)))
bias = bias[['feature', 'mod']].set_index('feature').iloc[:,0]

bias
```

```
Out[43]: feature
wins                0.800000
goals               0.843995
assists             0.854627
shots_on_goal       0.924914
shutouts            0.933702
powerplay_points     0.942256
goals_against_average 0.965344
plus_minus          0.980472
saves               0.987386
hits                0.991053
save_percentage      0.997072
blocks              1.000000
Name: mod, dtype: float64
```

```
In [45]: df[list(bias.keys())] *= bias  
  
df.head()
```

Out[45]:

	name	position	adp	goals	assists	plus_minus	powerplay_po
0	Connor McDavid	C	1.0	0.676725	0.854627	0.745444	0.443085
1	Nikita Kucherov	RW	2.0	0.697561	0.695209	0.822599	0.639388
2	Alex Ovechkin	LW	3.0	0.843995	0.481342	0.815956	0.471128
3	Sidney Crosby	C	4.0	0.601156	0.766055	0.762129	0.644997
4	Brad Marchand	LW	5.0	0.645052	0.649880	0.859441	0.448693

```
In [46]: from copy import deepcopy

cats = deepcopy(CATEGORIES)

cats.remove('goals')
cats.remove('shutouts')

df['score'] = df[cats].sum(axis=1)

df[['name', 'position', 'score']].head(10)
```

Out[46]:

	name	position	score
0	Connor McDavid	C	4.125333
1	Nikita Kucherov	RW	4.229589
2	Alex Ovechkin	LW	4.372368
3	Sidney Crosby	C	4.374385
4	Brad Marchand	LW	3.986886
5	Patrik Laine	RW	3.860518
6	Patrick Kane	RW	3.650039
7	Nathan MacKinnon	C	3.878440
8	John Tavares	C	4.006343
9	Auston Matthews	C	4.005856

Roster Positions:	C, C, LW, LW, RW, RW, D, D, D, D, G, G, BN, BN, BN, BN, IR, IR+
Send unjoined players email reminders:	Yes
Forwards/Defenseemen Stat Categories:	Goals (G), Assists (A), Plus/Minus (+/-), Powerplay Points (PPP), Shots on Goal (SOG), Hits (HIT), Blocks (BLK)
Goaltenders Stat Categories:	Wins (W), Goals Against Average (GAA), Saves (SV), Save Percentage (SV%), Shutouts (SHO)

```
In [47]: starters = {'C': 2, 'LW': 2, 'RW': 2, 'D': 4, 'G': 2}

players = sum(starters.values())
skaters = sum([value for key, value in starters.items() if key != 'G'])
goalies = players - skaters

print(skaters)
print(goalies)
```

```
10
2
```

```
In [48]: # df['score'] = df['score'] / players
df['score'] = np.where(df['position'] == 'G', df['score'] / goalies, df['score'] /
    skaters)

df[['name', 'position', 'score']].head()
```

Out[48]:

	name	position	score
0	Connor McDavid	C	0.412533
1	Nikita Kucherov	RW	0.422959
2	Alex Ovechkin	LW	0.437237
3	Sidney Crosby	C	0.437438
4	Brad Marchand	LW	0.398689

```
In [49]: raw.groupby('position').mean()
```

Out[49]:

	adp	goals	assists	plus_minus	powerplay_points
position					
C	90.814815	27.847870	41.831867	6.499907	16.208333
D	106.727273	11.287386	35.609716	3.702670	14.823864
G	71.807692	0.000000	0.000000	0.000000	0.000000
LW	81.846154	28.906122	36.666987	5.528686	13.282051
RW	107.931034	27.010718	35.137500	2.843103	15.698276

```
In [50]: pool_size = 10

for position, slots in starters.items():
    replacement = (
        df[df['position'] == position]
        .sort_values('score', ascending=False)
        .head(slots * pool_size)
        ['score']
        .mean()
    )
    df.loc[df['position'] == position, 'score'] = df['score'] - replacement
```



```
In [52]: df[['name', 'position', 'score']].sort_values('score', ascending=False).head()
```

Out[52]:

	name	position	score
11	Andrei Vasilevskiy	G	0.120399
2	Alex Ovechkin	LW	0.079319
29	Erik Karlsson	D	0.077475
19	Brent Burns	D	0.077302
17	Pekka Rinne	G	0.075867

```
In [53]: scale = blotto

df['score'] = df[['score']].apply(lambda x: scale(x, (0, 1)))
df['my_rank'] = df['score'].rank(method='average', ascending=False)
df = df.sort_values('my_rank')
```

```
In [54]: df['position_rank'] = df.groupby(['position'])['score'].rank(ascending=False)
df['arbitrage'] = df['adp'] - df['my_rank']
```

```
In [55]: df[['name', 'position', 'score', 'adp', 'my_rank', 'position_rank', 'arbitrage']].head()
```

Out[55]:

	name	position	score	adp	my_rank	position_rank	arbitrage
11	Andrei Vasilevskiy	G	1.000000	12.0	1.0	1.0	11.0
2	Alex Ovechkin	LW	0.929064	3.0	2.0	1.0	1.0
29	Erik Karlsson	D	0.925880	30.0	3.0	1.0	27.0
19	Brent Burns	D	0.925582	20.0	4.0	2.0	16.0
17	Pekka Rinne	G	0.923104	18.0	5.0	2.0	13.0

Round Team

Round 1			Round 2			Round 3		
1.	Connor McDavid (Edm - C)	Bee-ware🐝	1.	Jamie Benn (Dal - C,LW)	Rizwan & Isa...	1.	Braden Holtby (Was - G)	Bee-ware🐝
2.	Nikita Kucherov (TB - RW)	Camel Toews	2.	Patrick Kane (Chi - RW)	Candy Kanes	2.	Victor Hedman (TB - D)	Camel Toews
3.	Alex Ovechkin (Was - LW)	#FreeWilly	3.	Blake Wheeler (Wpg - C,RW)	fantasy.py	3.	Taylor Hall (NJ - LW)	#FreeWilly
4.	Auston Matthews (Tor - C)	Absolute Units	4.	Steven Stamkos (TB - C,RW)	Big Chief on...	4.	Claude Giroux (Phi - C,LW)	Absolute Units
5.	Sidney Crosby (Pit - C)	csm's Splend...	5.	Nathan MacKinnon (Col - C)	Pierre Snipes	5.	Sergei Bobrovsky (Cls - G)	csm's Splend...
6.	Patrik Laine (Wpg - RW)	Pierre Snipes	6.	Erik Karlsson (SJ - D)	csm's Splend...	6.	Connor Hellebuyck (Wpg - G)	Pierre Snipes
7.	John Tavares (Tor - C)	Big Chief on...	7.	Brent Burns (SJ - D)	Absolute Units	7.	Drew Doughty (LA - D)	Big Chief on...
8.	Andrei Vasilevskiy (TB - G)	fantasy.py	8.	David Pastrnak (Bos - RW)	#FreeWilly	8.	Dustin Byfuglien (Wpg - D)	fantasy.py
9.	Evgeni Malkin (Pit - C)	Candy Kanes	9.	Pekka Rinne (Nsh - G)	Camel Toews	9.	Tyler Seguin (Dal - C)	Candy Kanes
10.	Brad Marchand (Bos - LW)	Rizwan & Isa...	10.	Tuukka Rask (Bos - G)	Bee-ware🐝	10.	Roman Josi (Nsh - D)	Rizwan & Isa...
Round 4			Round 5			Round 6		
1.	Evgeny Kuznetsov (Was - C)	Rizwan & Isa...	1.	Martin Jones (SJ - G)	Bee-ware🐝	1.	Nikolaj Ehlers (Wpg - LW,RW)	Rizwan & Isa...
2.	Jonathan Quick (LA - G)	Candy Kanes	2.	Marc-Andre Fleury (VGK - G)	Camel Toews	2.	Evander Kane (SJ - LW)	Candy Kanes
3.	John Gibson (Anh - G)	fantasy.py	3.	Jack Eichel (Buf - C)	#FreeWilly	3.	Jakub Voracek (Phi - RW)	fantasy.py
4.	Frederik Andersen (Tor - G)	Big Chief on...	4.	Filip Forsberg (Nsh - LW)	Absolute Units	4.	Rickard Rakell (Anh - C,LW,RW)	Big Chief on...
5.	Mitchell Marner (Tor - C,RW)	Pierre Snipes	5.	Joe Pavelski (SJ - C,RW)	csm's Splend...	5.	Matt Murray (Pit - G)	Pierre Snipes
6.	Leon Draisaitl (Edm - C,RW)	csm's Splend...	6.	P.K. Subban (Nsh - D)	Pierre Snipes	6.	John Klingberg (Dal - D)	csm's Splend...

Round	Pick	Player	Position
1.	(8)	Andrei Vasilevskiy	G
2.	(13)	Blake Wheeler	C,RW
3.	(28)	Dustin Byfuglien	D
4.	(33)	John Gibson	G
5.	(48)	Artemi Panarin	LW
6.	(53)	Jakub Voracek	RW
7.	(68)	Torey Krug	D
8.	(73)	Nicklas Backstrom	C
9.	(88)	Patric Hornqvist	RW
10.	(93)	Ivan Provorov	D
11.	(108)	Reilly Smith	LW,RW
12.	(113)	Ryan Suter	D
13.	(128)	Colton Parayko	D
14.	(133)	T.J. Oshie	RW
15.	(148)	Dustin Brown	RW
16.	(153)	Boone Jenner	C,LW

[View All](#)

Projected Stats ▾

This table shows cumulative stats for all teams based on players they have drafted.

Rank	Team	G	A	+/-	PPP	SOG	HIT	BLK	W	GAA	SV	SV%	SHO	Total▾
1	fantasy.py	255.0	480.0	125.0	209.0	2771.0	1407.0	962.0	67.0	138.27	3314.0	.922	9.0	80
2	Johnny's Jets	305.0	512.0	28.0	270.0	2826.0	929.0	623.0	85.0	143.29	4708.0	.918	9.0	78
3	Rizwan & Isaac FTW	304.0	472.0	101.0	237.0	2843.0	1343.0	766.0	60.0	142.52	3346.0	.918	7.0	74
4	Dexter's Team	324.0	512.0	58.0	275.0	3050.0	810.0	934.0	57.0	144.65	3192.0	.918	6.0	72.5
5	The Mike Badcocks	309.0	498.0	103.0	232.0	2771.0	1121.0	822.0	62.0	147.62	3278.0	.916	7.0	66.5
6	Mack's Team	333.0	537.0	94.0	268.0	2891.0	841.0	647.0	55.0	145.41	3143.0	.916	7.0	65.5
7	Subban Hussein	284.0	498.0	111.0	231.0	2628.0	725.0	702.0	76.0	145.16	4021.0	.918	9.0	65
8	Michael-Anthony	282.0	445.0	46.0	230.0	2391.0	619.0	545.0	97.0	143.10	5027.0	.918	11.0	56.5
9	Big Chief on Campus	282.0	448.0	90.0	200.0	2503.0	1064.0	774.0	84.0	148.39	4720.0	.915	9.0	54.5
10	csm's Splendid Team	302.0	535.0	80.0	248.0	2785.0	871.0	768.0	51.0	149.77	2980.0	.915	5.0	47.5

< Week 4 > Matchup Totals Mon, Oct 29 Tue, Oct 30 Wed, Oct 31 Thu, Nov 1 Fri, Nov 2 Sat, Nov 3 Sun, Nov 4

fantasy.py
Max
28-13-7 | 1st



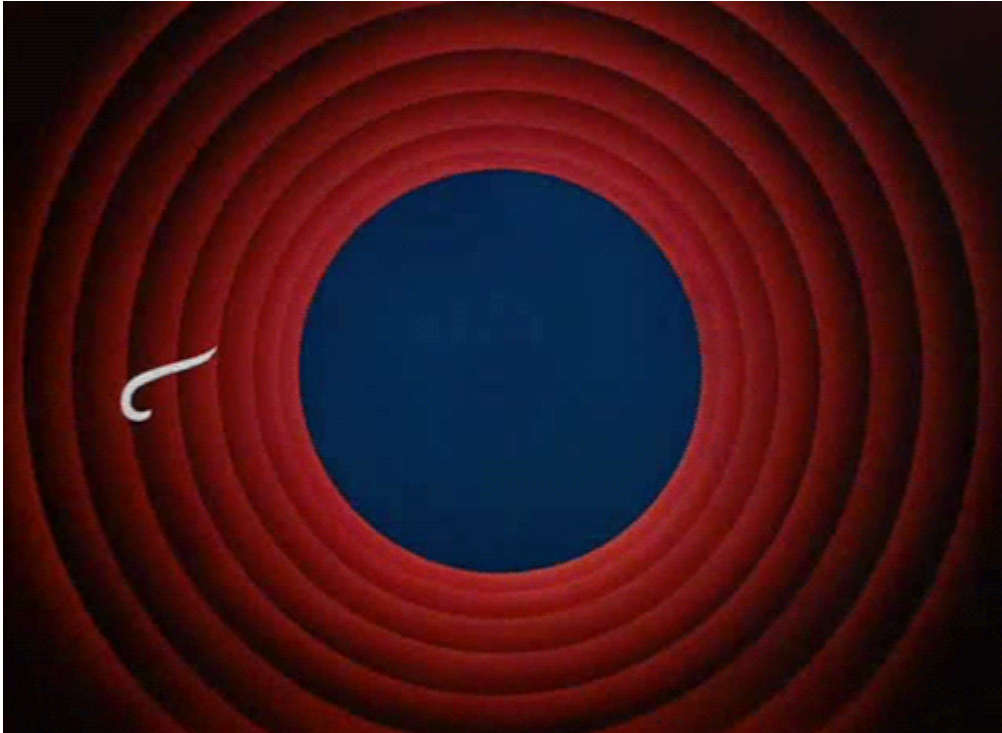
9 vs. 2
[Compare Managers](#)



Pierre Snipes
Kyle
13-30-5 | 10th

Team	G	A	+/-	PPP	SOG	HIT	BLK	W	GA*	GAA	SV	SA*	SV%	SHO	Score
fantasy.py	9	22	-2	14	117	54	37	3	11	2.18	127	138	.920	0	9
Pierre Snipes	19	14	1	10	116	32	30	1	18	3.77	118	136	.868	0	2

*Non-scoring stats



twitter: @maxhumber linkedin: /in/maxhumber email: maxhumber@gmail.com