UFC RECOMMENDATION ENGINE

Reading the Data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import datasets, linear_model
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import mean_squared_error, r2_score
import os
os.chdir("../DataFrames")

df = pd.read_csv("data.csv")
df.head(5)
```

Out[1]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak	..
0	Henry Cejudo	Marlon Moraes	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0	..
1	Valentina Shevchenko	Jessica Eye	Robert Madrigal	2019-06-08	Chicago, Illinois, USA	Red	True	Women's Flyweight	5	0.0	..
2	Tony Ferguson	Donald Cerrone	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0	..
3	Jimmie Rivera	Petr Yan	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	0.0	..
4	Tai Tuivasa	Blagoy Ivanov	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	0.0	..

5 rows × 145 columns

Missing Stances and Referees

In [2]:
```
# Dropping unnecessary rows and filling in values
```

In [3]:
```
# Show number of matches in dataset before removing matches with missing stance information
print('Number of matches prior to filtering: ' + str(len(df)))

# Remove matches with missing stance information
filter1 = df[df['B_Stance'].notnull()]
filter2 = filter1[filter1['R_Stance'].notnull()]
filter3 = filter2[filter2['Referee'].notnull()]
df = filter3
print('Number of matchs after filtering: ' + str(len(filter3)))
```

```
Number of matches prior to filtering: 5144
Number of matchs after filtering: 4865
```

Missing Numerical Data

In [4]:
```
df.columns[df.isnull().any()]
```

Out[4]:
```
Index(['B_avg_BODY_att', 'B_avg_BODY_landed', 'B_avg_CLINCH_att',
       'B_avg_CLINCH_landed', 'B_avg_DISTANCE_att', 'B_avg_DISTANCE_landed',
       'B_avg_GROUND_att', 'B_avg_GROUND_landed', 'B_avg_HEAD_att',
       'B_avg_HEAD_landed',
       ...
       'R_avg_opp_SUB_ATT', 'R_avg_opp_TD_att', 'R_avg_opp_TD_landed',
       'R_avg_opp_TD_pct', 'R_avg_opp_TOTAL_STR_att',
       'R_avg_opp_TOTAL_STR_landed', 'R_total_time_fought(seconds)',
       'R_Reach_cms', 'B_age', 'R_age'],
      dtype='object', length=104)
```

In [5]:
```
people = df[df['R_age'].isnull()].head(5)
people
```

Out[5]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak
4171	Per Eklund	Samy Schiavo	Leon Roberts	2008-10-18	Birmingham, England, United Kingdom	Red	False	Lightweight	3	1.0
4376	Jess Liaudin	Anthony Torres	Mario Yamasaki	2007-09-08	London, England, United Kingdom	Red	False	Welterweight	3	0.0
4438	Jess Liaudin	Dennis Siver	Steve Mazzagatti	2007-04-21	Manchester, England, United Kingdom	Red	False	Welterweight	3	0.0
4767	Keith Rockel	Chris Liguori	John McCarthy	2003-11-21	Uncasville, Connecticut, USA	Red	False	Middleweight	3	0.0
4908	Ben Earwood	Chris Lytle	Mario Yamasaki	2000-11-17	Atlantic City, New Jersey, USA	Red	False	Welterweight	2	0.0

5 rows × 145 columns

In [6]:
```
df['R_age'].median()
```

Out[6]:
```
29.0
```

In [7]:
```
df = df.fillna(df.median())
df.loc[people.index.tolist()]
```

Out[7]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak
4171	Per Eklund	Samy Schiavo	Leon Roberts	2008-10-18	Birmingham, England, United Kingdom	Red	False	Lightweight	3	1.0
4376	Jess Liaudin	Anthony Torres	Mario Yamasaki	2007-09-08	London, England, United Kingdom	Red	False	Welterweight	3	0.0
4438	Jess Liaudin	Dennis Siver	Steve Mazzagatti	2007-04-21	Manchester, England, United Kingdom	Red	False	Welterweight	3	0.0
4767	Keith Rockel	Chris Liguori	John McCarthy	2003-11-21	Uncasville, Connecticut, USA	Red	False	Middleweight	3	0.0
4908	Ben Earwood	Chris Lytle	Mario Yamasaki	2000-11-17	Atlantic City, New Jersey, USA	Red	False	Welterweight	2	0.0

5 rows × 145 columns

Adding the Gender column

In [8]:
```
view = df
#dropping duplicates
newView = view.drop_duplicates(subset = "B_fighter", keep = "first")

newView.head()
```

Out[8]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak	..
0	Henry Cejudo	Marlon Moraes	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0	..
1	Valentina Shevchenko	Jessica Eye	Robert Madrigal	2019-06-08	Chicago, Illinois, USA	Red	True	Women's Flyweight	5	0.0	..
2	Tony Ferguson	Donald Cerrone	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0	..
3	Jimmie Rivera	Petr Yan	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	0.0	..
4	Tai Tuivasa	Blagoy Ivanov	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	0.0	..

5 rows × 145 columns

In [9]:
```
filtered_df_female_new  = newView.loc[(newView['weight_class'] == "Women's Bantamweight") | (ne
```

In [10]:
```
filtered_df_male_new  = newView.loc[(newView['weight_class'] == "Bantamweight") | (newView['wei
```

In [11]:
```
#female 0, male 1
filtered_df_female_new["Gender"]=0
filtered_df_male_new["Gender"]=1
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [12]:
```
frames_new=[filtered_df_male_new,filtered_df_female_new]
```

In [13]:
```
viewWithDuplicates=pd.concat(frames_new)
```

In [14]:
```
#Males = 1
#Females = 0
finalView = viewWithDuplicates.drop_duplicates(subset = "B_fighter", keep = "first")
#dropping duplicates
finalView
```

In [14]:
```
#Males = 1
#Females = 0
finalView = viewWithDuplicates.drop_duplicates(subset = "B_fighter", keep = "first")
#dropping duplicates
finalView
```

Out[14]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak
0	Henry Cejudo	Marlon Moraes	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0
2	Tony Ferguson	Donald Cerrone	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0
3	Jimmie Rivera	Petr Yan	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	0.0
4	Tai Tuivasa	Blagoy Ivanov	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	0.0
6	Aljamain Sterling	Pedro Munhoz	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	False	Bantamweight	3	0.0
...	...	...	...	...	...	...	...	...	...	...
2073	Miesha Tate	Sara McMann	Marc Goddard	2015-01-31	Las Vegas, Nevada, USA	Red	False	Women's Bantamweight	3	0.0
2266	Jessica Andrade	Larissa Pacheco	Fernando Yamasaki	2014-09-13	Brasilia, Distrito Federal, Brazil	Red	False	Women's Bantamweight	3	0.0
2352	Claudia Gadelha	Tina Lahdemaki	Liam Kerrigan	2014-07-16	Atlantic City, New Jersey, USA	Red	False	Women's Strawweight	3	0.0
2798	Amanda Nunes	Sheila Gaff	Herb Dean	2013-08-03	Rio de Janeiro, Brazil	Red	False	Women's Bantamweight	3	1.0

In [15]:
```
finalView.to_csv("finalView.csv")
```

Dataframes for Specifically the Blue and Red Corner

In [16]:
```
blueDf = finalView[finalView.columns.drop(list(finalView.filter(regex='R')))]
redDf = finalView[finalView.columns.drop(list(finalView.filter(regex="B")))]
```

In [17]:
```
blueDf.to_csv("blue.csv")
redDf.to_csv("red.csv")
```

Dataframes for Specifically the Blue and Red stats

In [18]:
```
#dropping red and blue Referee, date, location, Winner, title_bout, weight_class, and the fight
#note that this is used for kNN as attributes of a fighter, so you can disregard the fighter hi
#since that fighter will coorespond with the row anyway
#another thing is that you can always just modify blueDf
```

In [19]:
```
blueStats = blueDf.drop(["B_fighter","date", "location", "Winner", "title_bout","weight_class"]
```

In [20]:
```
redStats = redDf.drop(["R_fighter","date", "location", "Winner", "title_bout","weight_class"],
```

In [21]:
```
blueStats.to_csv("blueStats.csv")
redStats.to_csv("redStats.csv")
```

Dataframes Separating Men and Women

In [22]:
```
#For men
```

In [23]:
```
menView = finalView.loc[(finalView['Gender'] == 1)]
menView.to_csv("menView.csv")
```

In [24]:
```
#for women
womenView = finalView.loc[(finalView["Gender"]==0)]
womenView.to_csv("womenView.csv")
```

In [25]:
```
finalView
```

Out[25]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak
0	Henry Cejudo	Marlon Moraes	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0
2	Tony Ferguson	Donald Cerrone	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0
3	Jimmie Rivera	Petr Yan	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	0.0
4	Tai Tuivasa	Blagoy Ivanov	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	0.0
6	Aljamain Sterling	Pedro Munhoz	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	False	Bantamweight	3	0.0
...	...	...	...	...	...	...	...	...	...	...
2073	Miesha Tate	Sara McMann	Marc Goddard	2015-01-31	Las Vegas, Nevada, USA	Red	False	Women's Bantamweight	3	0.0
2266	Jessica Andrade	Larissa Pacheco	Fernando Yamasaki	2014-09-13	Brasilia, Distrito Federal, Brazil	Red	False	Women's Bantamweight	3	0.0
2352	Claudia Gadelha	Tina Lahdemaki	Liam Kerrigan	2014-07-16	Atlantic City, New Jersey, USA	Red	False	Women's Strawweight	3	0.0
2798	Amanda Nunes	Sheila Gaff	Herb Dean	2013-08-03	Rio de Janeiro, Brazil	Red	False	Women's Bantamweight	3	1.0
2830	Alexis Davis	Rosi Sexton	Herb Dean	2013-06-15	Winnipeg, Manitoba, Canada	Red	False	Women's Bantamweight	3	0.0

1662 rows × 146 columns

SVD

In [1]:
```
import pandas as pd
import numpy as np
import os
from surprise import SVD, Dataset, Reader
os.chdir("../DataFrames")
df = pd.read_csv("finalView.csv",index_col=0)
blue = pd.read_csv("blue.csv", index_col = 0)
red = pd.read_csv("red.csv", index_col = 0 )
df.head(5)
```

Out[1]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak	...	R_win_by_Submission	R_win_by_
0	Henry Cejudo	Marlon Moraes	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0	...	0.0	
2	Tony Ferguson	Donald Cerrone	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0	...	6.0	
3	Jimmie Rivera	Petr Yan	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	0.0	...	0.0	
4	Tai Tuivasa	Blagoy Ivanov	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	0.0	...	0.0	
6	Aljamain Sterling	Pedro Munhoz	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	False	Bantamweight	3	0.0	...	3.0	

5 rows × 146 columns

In [2]:
```
#For the Blue Corner
```

In [3]:
```
newView = blue
```

In [4]:
```
#Creating the user class which allows for implicit feedback
class User:
    #Constructor for the user class
    def __init__(self, username, liked, disliked):
        self.username = username
        self.liked = liked
        self.disliked = disliked

    #getters
    def get_username(self):
        return self.username

    def get_liked(self):
        return self.liked

    def get_disliked(self):
        return self.disliked

    #Gets the rating from the liked or disliked.
def get_rating(name, liked, disliked):
    if name in liked:
        return 5
    elif name in disliked:
        return 1

    #Trains the dataset
def get_trainset(users):
    all_data = []
    for user in users:
        user_all_fighters = [x for y in [user.get_liked(), user.get_disliked()] for x in y]
        filtered_new_view = newView[newView.B_fighter.isin(user_all_fighters)].copy()
        ratings = filtered_new_view.apply(lambda row: get_rating(row.B_fighter, user.get_liked(), user.get_disliked()),
        fighter_names = filtered_new_view['B_fighter']
        for fighter, rate in zip(fighter_names, ratings):
            all_data.append([user.get_username(), fighter, rate])
    tdf = pd.DataFrame(all_data)
    reader = Reader(rating_scale=(1, 5))
    train_set = Dataset.load_from_df(tdf, reader).build_full_trainset()
    return train_set
```

```
#implementation of SVD which would allow for
def get_top_3_recommendations(users, recommends_user):
    all_fighters = []
    for user in users:
        all_fighters.extend(user.get_liked())
        all_fighters.extend(user.get_disliked())
    all_fighters = np.unique(all_fighters)
    train_set = get_trainset(users)
    algorithm = SVD()
    algorithm.fit(train_set)
    predictions = []
    for fighter in all_fighters:
        rating = algorithm.predict(user, fighter)
        if not(fighter in recommends_user.get_liked() or fighter in recommends_user.get_dislike
            predictions.append((fighter, rating.est))
    sort_data = sorted(predictions, key=lambda x: x[1], reverse=True)
    return sort_data[:3]


user1 = User("Alex", ["Khabib Nurmagomedov", "Amanda Nunes", "Kamaru Usman", "Weili Zhang", "De
              ["Colby Covington", "Ronda Rousey", "Tony Ferguson", "Marlon Moraes"])
user2 = User("Diel", ["Colby Covington", "Jorge Masvidal", "Carlos Condit", "Anderson Silva"],
user3 = User("Joe", ["Junior dos Santos", "Josh Barnett", "Garreth McLellan"], ["Damian Grabows
user4 = User("Morgan", ["Colby Covington"], ["Amanda Nunes"])
get_top_3_recommendations([user1, user2, user3, user4], user1)
```

Out[4]:
```
[('Garreth McLellan', 3.5493907458054514),
 ('Josh Barnett', 3.5386405548168063),
 ('Jorge Masvidal', 3.519973745030947)]
```

In [5]:
```
#For the Red Corner with the same input
```

In [6]:
```
newView = red
```

In [7]:
```
#Creating the user class which allows for implicit feedback
class User:
    #Constructor for the user class
    def __init__(self, username, liked, disliked):
        self.username = username
        self.liked = liked
        self.disliked = disliked

    #getters
    def get_username(self):
        return self.username

    def get_liked(self):
        return self.liked

    def get_disliked(self):
        return self.disliked

    #Gets the rating from the liked or disliked.
def get_rating(name, liked, disliked):
    if name in liked:
        return 5
    elif name in disliked:
        return 1

    #Trains the dataset
def get_trainset(users):
    all_data = []
    for user in users:
        user_all_fighters = [x for y in [user.get_liked(), user.get_disliked()] for x in y]
        filtered_new_view = newView[newView.R_fighter.isin(user_all_fighters)].copy()
        ratings = filtered_new_view.apply(lambda row: get_rating(row.R_fighter, user.get_liked(), user.get_disliked()),
        fighter_names = filtered_new_view['R_fighter']
        for fighter, rate in zip(fighter_names, ratings):
            all_data.append([user.get_username(), fighter, rate])
    tdf = pd.DataFrame(all_data)
    reader = Reader(rating_scale=(1, 5))
    train_set = Dataset.load_from_df(tdf, reader).build_full_trainset()
    return train_set
```

```
#implementation of SVD which would allow for
def get_top_3_recommendations(users, recommends_user):
    all_fighters = []
    for user in users:
        all_fighters.extend(user.get_liked())
        all_fighters.extend(user.get_disliked())
    all_fighters = np.unique(all_fighters)
    train_set = get_trainset(users)
    algorithm = SVD()
    algorithm.fit(train_set)
    predictions = []
    for fighter in all_fighters:
        rating = algorithm.predict(user, fighter)
        if not(fighter in recommends_user.get_liked() or fighter in recommends_user.get_disliked()):
            predictions.append((fighter, rating.est))
    sort_data = sorted(predictions, key=lambda x: x[1], reverse=True)
    return sort_data[:3]


user1 = User("Alex", ["Khabib Nurmagomedov", "Amanda Nunes", "Kamaru Usman", "Weili Zhang", "Derrick Lewis"],
             ["Colby Covington", "Ronda Rousey", "Tony Ferguson", "Marlon Moraes"])
user2 = User("Diel", ["Colby Covington", "Jorge Masvidal", "Carlos Condit", "Anderson Silva"], ["Yair Rodriguez", "Rick
user3 = User("Joe", ["Junior dos Santos", "Josh Barnett", "Garreth McLellan"], ["Damian Grabowski", "Mark Coleman"])
user4 = User("Morgan", ["Colby Covington"], ["Amanda Nunes"])
get_top_3_recommendations([user1, user2, user3, user4], user1)
```

Out[7]:
```
[('Carlos Condit', 3.867842143504462),
 ('Anderson Silva', 3.8291727024148985),
 ('Josh Barnett', 3.8246471508329014)]
```

PCA Implementation

We apply dimensionality reduction to obtain an ordered list of components that account for the largest variance in the data set in order to ultimately group similar fighters based on their fighting styles

In [3]:
```
df_new=df.drop(columns=["R_fighter","B_fighter","Referee","date","location","Winner","title_bou
#pd.set_option('display.max_rows', 999)
```

Then we find the percentage of variance explained by each of the selected components.

In [4]:
```
from sklearn import (cluster, datasets, decomposition, ensemble, manifold, random_projection)

pca = decomposition.PCA(n_components=8)
X_pca=pca.fit_transform(df_new)
X_pca
print(pca.explained_variance_ratio_)
```

```
[0.50005302 0.35087945 0.04506566 0.03209157 0.01925669 0.01268431
 0.01145262 0.00457093]
```

In [5]:
```
df.keys()
```

Out[5]:
```
Index(['R_fighter', 'B_fighter', 'Referee', 'date', 'location', 'Winner',
       'title_bout', 'weight_class', 'no_of_rounds', 'B_current_lose_streak',
       ...
       'R_win_by_Submission', 'R_win_by_TKO_Doctor_Stoppage', 'R_wins',
       'R_Stance', 'R_Height_cms', 'R_Reach_cms', 'R_Weight_lbs', 'B_age',
       'R_age', 'Gender'],
      dtype='object', length=146)
```

In [6]:
```
len(X_pca)
```

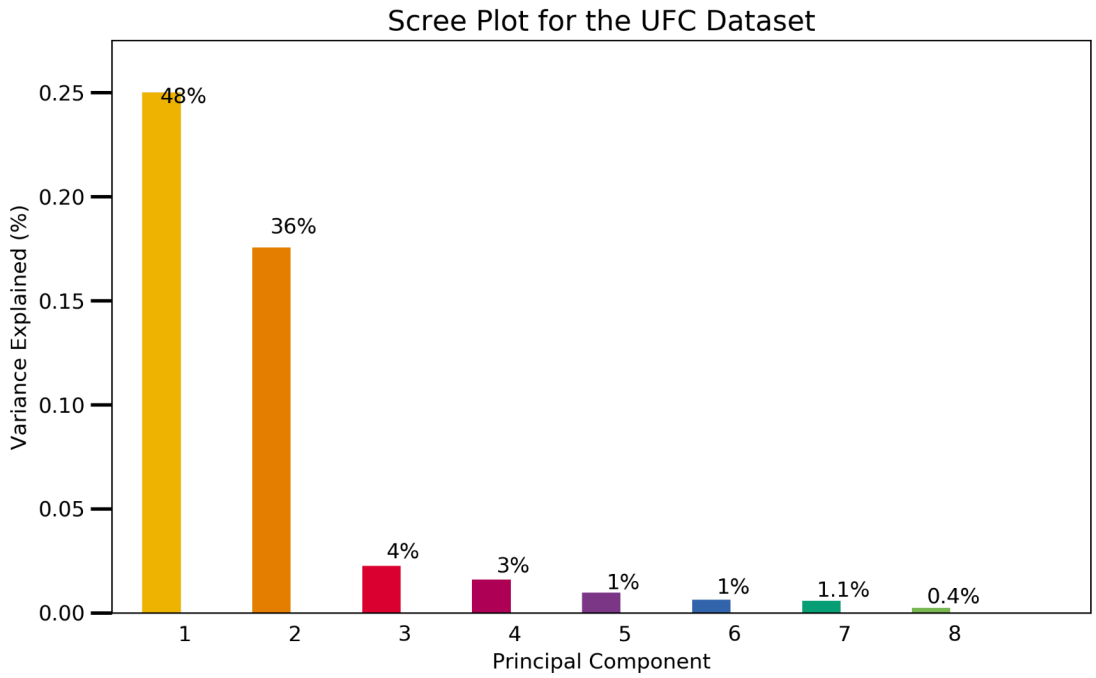
Out[6]:
```
1662
```

In [7]:

```
import numpy as np
import pylab as pl
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from mpl_toolkits.mplot3d import Axes3D
# Generate scree plot

N = 8
ind = np.arange(N)  # the x locations for the groups

vals = [0.48580999,
        0.3601545,
        0.04796335,
        0.03418453,
        0.01828115,
        0.01379265,
        0.01128048,
        0.00454691]


pl.figure(figsize=(10, 6), dpi=250)
ax = pl.subplot(111)
ax.bar(ind, pca.explained_variance_ratio_, 0.35,
       color=[(0.949, 0.718, 0.004),
              (0.898, 0.49, 0.016),
              (0.863, 0, 0.188),
              (0.694, 0, 0.345),
              (0.486, 0.216, 0.541),
              (0.204, 0.396, 0.667),
              (0.035, 0.635, 0.459),
              (0.486, 0.722, 0.329),
             ])

ax.annotate(r"%d%%" % (int(vals[0]*100)), (ind[0]+0.2, vals[0]), va="bottom", ha="center", fontsize=12)
ax.annotate(r"%d%%" % (int(vals[1]*100)), (ind[1]+0.2, vals[1]), va="bottom", ha="center", fontsize=12)
ax.annotate(r"%d%%" % (int(vals[2]*100)), (ind[2]+0.2, vals[2]), va="bottom", ha="center", fontsize=12)
ax.annotate(r"%d%%" % (int(vals[3]*100)), (ind[3]+0.2, vals[3]), va="bottom", ha="center", fontsize=12)
ax.annotate(r"%d%%" % (int(vals[4]*100)), (ind[4]+0.2, vals[4]), va="bottom", ha="center", fontsize=12)
ax.annotate(r"%d%%" % (int(vals[5]*100)), (ind[5]+0.2, vals[5]), va="bottom", ha="center", fontsize=12)
ax.annotate(r"%s%%" % ((str(vals[6]*100)[:4 + (0-1)])), (ind[6]+0.2, vals[6]), va="bottom", ha="center", fontsize=12)
ax.annotate(r"%s%%" % ((str(vals[7]*100)[:4 + (0-1)])), (ind[7]+0.2, vals[7]), va="bottom", ha="center", fontsize=12)
```

```
ax.set_xticklabels(('       0',
                    '       1',
                    '       2',
                    '       3',
                    '       4',
                    '       5',
                    '       6',
                    '       7',
                    '       8'),
                   fontsize=12)
ax.set_yticklabels(('0.00', '0.05', '0.10', '0.15', '0.20', '0.25','0.35','0.45','0.55'), fonts
ax.set_ylim(0, .55)
ax.set_xlim(0-0.45, 8+0.45)

ax.xaxis.set_tick_params(width=0)
ax.yaxis.set_tick_params(width=2, length=12)

ax.set_xlabel("Principal Component", fontsize=12)
ax.set_ylabel("Variance Explained (%)", fontsize=12)

pl.title("Scree Plot for the UFC Dataset", fontsize=16)
```

Out[7]: Text(0.5, 1.0, 'Scree Plot for the UFC Dataset')



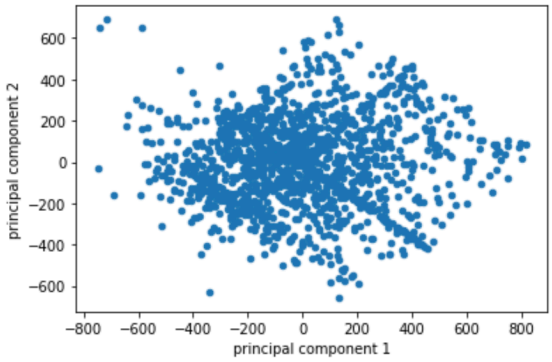
From the scree plot it is clear that the elbow point is at 6 components so we proceed to use this number in our PCA analysis

In [8]:
```
pca_new = decomposition.PCA(n_components=2)
X_pca_new = pca_new.fit_transform(df_new)

principalDf = pd.DataFrame(data = X_pca_new
             , columns = ['principal component 1', 'principal component 2'])
```

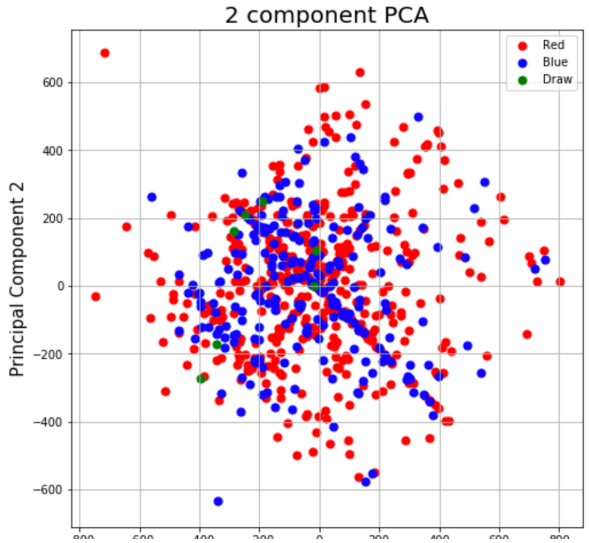
In [9]:
```
pd.set_option('display.max_rows', 6000)
pd.set_option('display.max_columns', 6000)
finalDf = pd.concat([principalDf, df[['R_fighter','B_fighter','Winner','title_bout']]], axis =
```

In [10]:
```
av=principalDf["principal component 1"]
bv=principalDf["principal component 2"]
principalDf.plot(kind='scatter', x="principal component 1", y="principal component 2")
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1a18bb33d0>

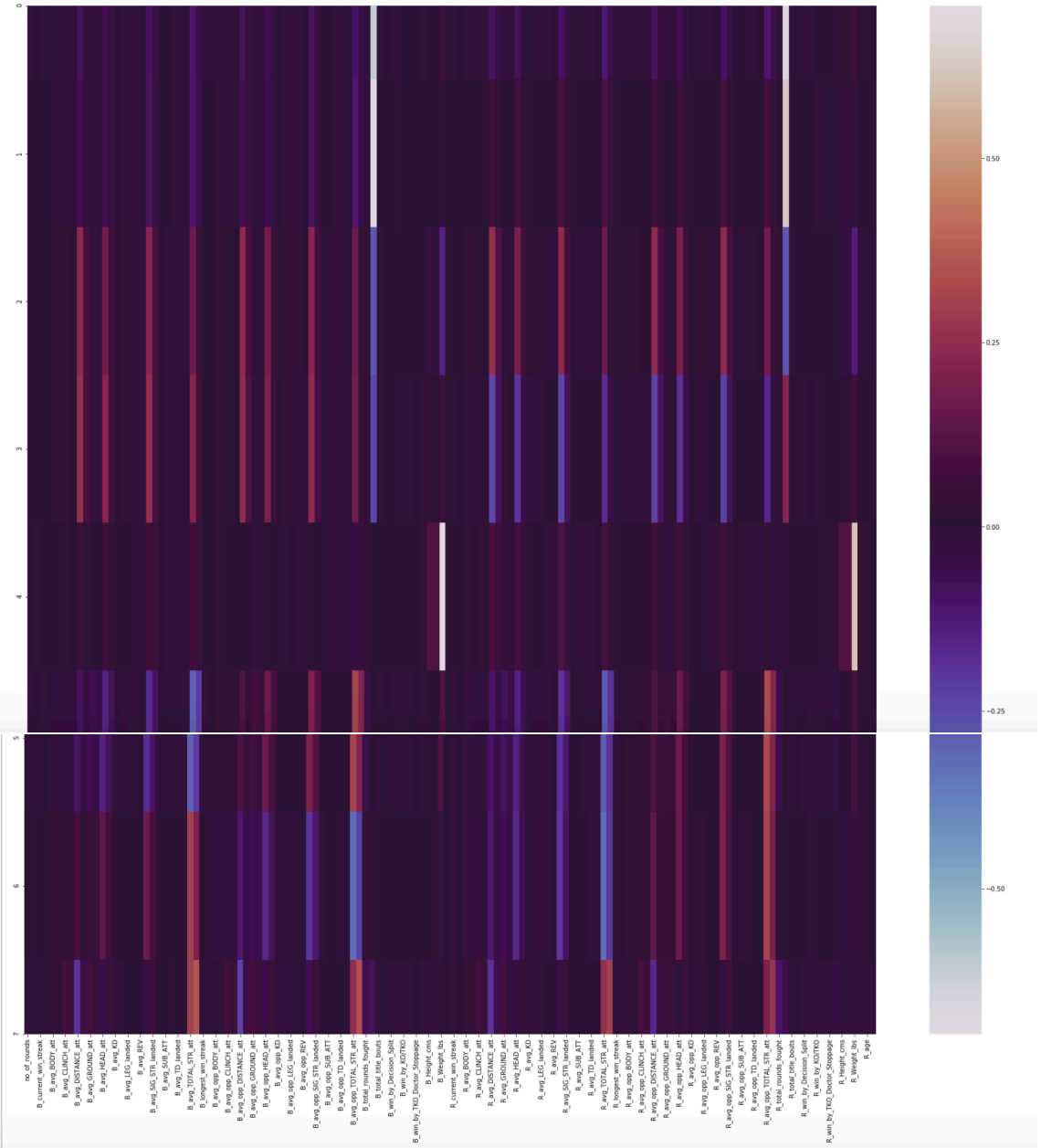


In [11]:
```
import matplotlib.pyplot as plt

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Red', 'Blue','Draw']
colors = ['r', 'b','g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Winner'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



In [12]:
```
import seaborn as sns

map = pd.DataFrame(pca.components_,columns=df_new.columns)
plt.figure(figsize=(30,30))
sns.heatmap(map,cmap='twilight')
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1de93710>

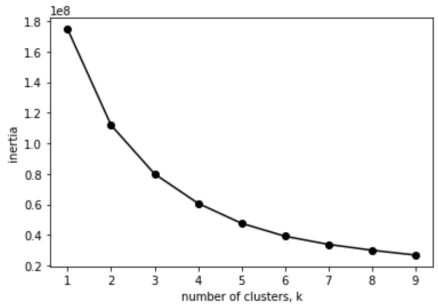


K-Means

In [1]:
```
import pandas as pd
import numpy as np
import os
import pylab as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from mpl_toolkits.mplot3d import Axes3D
from sklearn import (cluster, datasets, decomposition, ensemble, manifold, random_projection)

os.chdir("../DataFrames")
df = pd.read_csv('finalView.csv',  index_col=0)
```

In [2]:
```
df_new=df.drop(columns=["R_fighter","B_fighter","Referee","date","location","Winner","title_bout","weight_class","B_Sta
#pd.set_option('display.max_rows', 999)
pca_new = decomposition.PCA(n_components=2)
X_pca_new = pca_new.fit_transform(df_new)
#X_pca_new

principalDf = pd.DataFrame(data = X_pca_new
             , columns = ['principal component 1', 'principal component 2'])
```

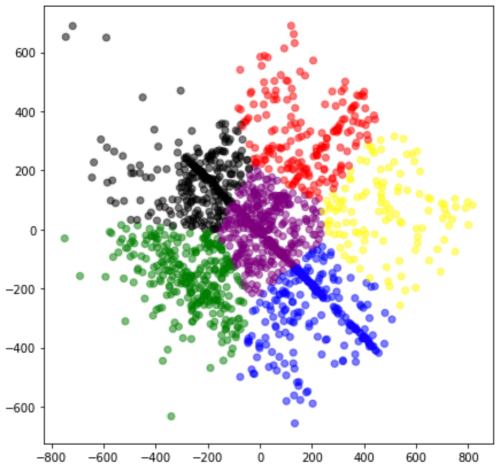
In [3]:
```
from sklearn.cluster import KMeans
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(principalDf.iloc[:,:2])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



It is clear that the elbow point is 6

In [5]:
```
pca_new_2 = decomposition.PCA(n_components=2)
X_pca_new_2 = pca_new_2.fit_transform(df_new)
```

In [6]:
```
#Set a 2 KMeans clustering
kmeans = KMeans(n_clusters = 6)

#Compute cluster centers and predict cluster indices
X_clustered = kmeans.fit_predict(X_pca_new_2)

#Define our own color map
LABEL_COLOR_MAP = {0:'r', 1: 'g', 2: 'b', 3: 'yellow', 4: 'purple', 5: 'black'}
label_color = [LABEL_COLOR_MAP[l] for l in X_clustered]

# Plot the scatter digram
plt.figure(figsize = (7,7))
plt.scatter(X_pca_new_2[:,0],X_pca_new_2[:,1], c= label_color, alpha=0.5)
plt.show()
```



Alternative Least Squares ¶

The above method is used here to find similar fighters to a particular fighter based on a paritcular feature (in this case I chose avg takedown percentages). This method will also be used to recommend fighters based on that same feature.

In [2]:
```
#For the blue corner
```

In [3]:
```
import sys
import pandas as pd
import numpy as np
import scipy.sparse as sparse
from scipy.sparse.linalg import spsolve
import random
from sklearn.preprocessing import MinMaxScaler
import implicit
import os
os.chdir("../DataFrames")


#load dataset. Blue corner fighters (B_fighter) is used here but is also applicable to fighters
raw_data = pd.read_csv('finalView.csv',  index_col=0)

# Drop NaN columns
data_new = raw_data.dropna()
data_new= data_new.copy()

#Convert fighter name and weight class to a category dtype
data_new['weight_class_name'] = data_new["weight_class"].astype("category").cat.codes
data_new['B_fighter_name'] = data_new["B_fighter"].astype("category").cat.codes

##Enter fighter feature for recommendation. In this case i chose takedown percentages
fighter_feature=data_new["B_avg_TD_pct"]

#createing two matrices, one for finding similar fighters and one for best fighter recommendati
sparse_item_weight_class_name = sparse.csr_matrix((fighter_feature.astype(float), (data_new['B_
sparse_item_B_fighter_name = sparse.csr_matrix((fighter_feature.astype(float), (data_new['weigh
```

In [4]:
```
# Initialize the als model and fit it using the sparse item feature-fighter matrix
model = implicit.als.AlternatingLeastSquares(factors=20, regularization=0.1, iterations=20)
alpha_val = 40

# Calculate the data confidence by multiplying it by our alpha value.
data_conf = (sparse_item_weight_class_name * alpha_val).astype('double')

#Fit the model
model.fit(data_conf)
```

```
WARNING:root:OpenBLAS detected. Its highly recommend to set the environment variable 'export OPENBLAS_NUM_THREADS=1'
to disable its internal multithreading
```

```
HBox(children=(FloatProgress(value=0.0, max=20.0), HTML(value='')))
```

In [5]:
```
##get similar fighters based on the particular feature set above
def get_similar_fighters(fighter_name):
    #convert fighter name to index
    list1=data_new.index[data_new['B_fighter'] == fighter_name].tolist()
    fighter_id = list1[0]
    n_similar = 10

    # Use implicit to get similar fighters.
    similar = model.similar_items(fighter_id, n_similar)
    #Show names of similar fighters
    for item in similar:
        idx, score = item
        print (data_new.B_fighter.loc[data_new.B_fighter_name == idx].iloc[0])
```

In [6]:
```
get_similar_fighters("Tony Ferguson")
```

```
Phil Harris
Jenel Lausa
Chris Beal
Chris Cariaso
Jose Torres
Brandon Moreno
Alptekin Ozkilic
Marco Beltran
Henry Cejudo
Darren Uyenoyama
```

In [7]:
```
get_similar_fighters("Amanda Nunes")
```

```
Cameron Dollar
Mac Danzig
Paul Taylor
Michel Prazeres
Clay Guida
Melvin Guillard
Hermes Franca
Rodrigo Damm
Thomas Gifford
Danny Castillo
```

In [8]:
```
#Get best fighter recommendations based on selected feature above
def best_fighter_recommendations(weight_class_iter, sparse_fighter_content, fighter_vecs, weight_class_vecs, num_conter

    # Get the array scores from the sparse weight class matrix
    fighter_array = sparse_item_B_fighter_name[weight_class_id,:].toarray()
    # Gives a new shape to an array without changing its data.
    fighter_array= fighter_array.reshape(-1) + 1

    # Mark visited items in array to 0
    fighter_array[fighter_array > 1] = 0

    # Get dot product of fighter vector and all weight class vectors
    recommendation_vector = fighter_vecs[weight_class_id,:].dot(weight_class_vecs.T).toarray()

    # Scale this recommendation vector between 0 and 1
    min_max = MinMaxScaler()
    recommendation_vector_scaled = min_max.fit_transform(recommendation_vector.reshape(-1,1))[:,0]
    # Scaling already visted items in array
    recommendation_vector_final = fighter_array * recommendation_vector_scaled
    # Sort the indices of the weight class into order of best recommendations
    weight_class_iter = np.argsort(recommendation_vector_final)[::-1][:num_contents]

    #Empty list to store fighters and scores
    fighters = []
    scores = []

    for iter in weight_class_iter:
        # Append fighters and scores to the list
        fighters.append(data_new.B_fighter.loc[data_new.B_fighter_name == iter].iloc[0])
        scores.append(recommendation_vector_final[iter])

    recommendations = pd.DataFrame({'Fighter': fighters, 'score': scores})

    return recommendations
```

In [9]:
```
# Get the fighter and weight class vectors converting them to csr matrices
fighter_vecs = sparse.csr_matrix(model.user_factors)
weight_class_vecs = sparse.csr_matrix(model.item_factors)

# Create best recommendations based on feature selected above. Set to 0 for best recommendations
weight_class_id = 0

#Get recommendations
recommendations = best_fighter_recommendations(weight_class_id, sparse_item_B_fighter_name, fighter_vecs, weight_class_

print(recommendations)
```

	Fighter	score
0	Chris Gutierrez	0.056119
1	Marcos Vinicius	0.051951
2	Vince Morales	0.050364
3	Song Yadong	0.050241
4	Walel Watson	0.041870
5	Khalid Taha	0.039220
6	Tim Gorman	0.037402
7	Ian Loveland	0.036012
8	Damacio Page	0.034774
9	Ian Entwistle	0.031591

In [10]:
```
#For the red corner
```

In [11]:
```
raw_data = pd.read_csv('finalView.csv',  index_col=0)
# Drop NaN columns
data_new = raw_data.dropna()
data_new= data_new.copy()

#Convert fighter name and weight class to a category dtype
data_new['weight_class_name'] = data_new["weight_class"].astype("category").cat.codes
data_new['R_fighter_name'] = data_new["R_fighter"].astype("category").cat.codes

##Enter fighter feature for recommendation. In this case I chose takedown percentages
fighter_feature=data_new["R_avg_TD_pct"]

#createing two matrices, one for finding similar fighters and one for best fighter recommendations in each weight_class
sparse_item_weight_class_name = sparse.csr_matrix((fighter_feature.astype(float), (data_new['R_fighter_name'], data_new
sparse_item_R_fighter_name = sparse.csr_matrix((fighter_feature.astype(float), (data_new['weight_class_name'], data_new
```

In [12]:
```
# Initialize the als model and fit it using the sparse item feature-fighter matrix
model = implicit.als.AlternatingLeastSquares(factors=20, regularization=0.1, iterations=20)
alpha_val = 40

# Calculate the data confidence by multiplying it by our alpha value.
data_conf = (sparse_item_weight_class_name * alpha_val).astype('double')

#Fit the model
model.fit(data_conf)
```

```
HBox(children=(FloatProgress(value=0.0, max=20.0), HTML(value='')))
```

In [13]:
```
##get similar fighters based on the particular feature set above
def get_similar_fighters(fighter_name):
    #convert fighter name to index
    list1=data_new.index[data_new['R_fighter'] == fighter_name].tolist()
    fighter_id = list1[0]
    n_similar = 10

    # Use implicit to get similar fighters.
    similar = model.similar_items(fighter_id, n_similar)
    #Show names of similar fighters
    for item in similar:
        idx, score = item
        print (data_new.R_fighter.loc[data_new.R_fighter_name == idx].iloc[0])
```

In [14]:
```
get_similar_fighters("Tony Ferguson")
```

```
Abel Trujillo
Paul Sass
Guy Mezger
Gilbert Burns
Alex White
Joe Lauzon
Islam Makhachev
Desmond Green
Erik Koch
Roosevelt Roberts
```

In [15]:
```
get_similar_fighters("Amanda Nunes")
```

```
Matt Grice
Desmond Green
Stevie Ray
Callan Potter
Francisco Trevino
Alex White
Michael Chiesa
Gilbert Burns
Jim Miller
Rolando Delgado
```

In [16]:
```
get_similar_fighters("Khabib Nurmagomedov")
```

```
Jason Lambert
Corey Anderson
Anthony Perosh
Ryan Bader
Eliot Marshall
Volkan Oezdemir
Sam Hoger
Rashad Evans
Jason Brilz
Matt Hamill
```

In [17]:
```
#Get best fighter recommendations based on selected feature above
def best_fighter_recommendations(weight_class_iter, sparse_fighter_content, fighter_vecs, weight_class_vecs, num_conter

    # Get the array scores from the sparse weight class matrix
    fighter_array = sparse_item_R_fighter_name[weight_class_id,:].toarray()
    # Gives a new shape to an array without changing its data.
    fighter_array= fighter_array.reshape(-1) + 1

    # Mark visited items in array to 0
    fighter_array[fighter_array > 1] = 0

    # Get dot product of fighter vector and all weight class vectors
    recommendation_vector = fighter_vecs[weight_class_id,:].dot(weight_class_vecs.T).toarray()

    # Scale this recommendation vector between 0 and 1
    min_max = MinMaxScaler()
    recommendation_vector_scaled = min_max.fit_transform(recommendation_vector.reshape(-1,1))[:,0]
    # Scaling already visted items in array
    recommendation_vector_final = fighter_array * recommendation_vector_scaled
    # Sort the indices of the weight class into order of best recommendations
    weight_class_iter = np.argsort(recommendation_vector_final)[::-1][:num_contents]

    #Empty list to store fighters and scores
    fighters = []
    scores = []

    for iter in weight_class_iter:
        # Append fighters and scores to the list
        fighters.append(data_new.R_fighter.loc[data_new.R_fighter_name == iter].iloc[0])
        scores.append(recommendation_vector_final[iter])

    recommendations = pd.DataFrame({'Fighter': fighters, 'score': scores})

    return recommendations
```

In [18]:
```
# Get the fighter and weight class vectors converting them to csr matrices
fighter_vecs = sparse.csr_matrix(model.user_factors)
weight_class_vecs = sparse.csr_matrix(model.item_factors)

# Create best recommendations based on feature selected above. Set to 0 for best recommendation
weight_class_id = 0

#Get recommendations
recommendations = best_fighter_recommendations(weight_class_id, sparse_item_B_fighter_name, fig

print(recommendations)
```

```
          Fighter     score
0     Alex Caceres  0.230035
1     Brad Pickett  0.119081
2    Lucas Martins  0.081333
3   Edwin Figueroa  0.078986
4     Benito Lopez  0.068402
5   Johnny Eduardo  0.060121
6     Dustin Pague  0.056784
7       Chris Beal  0.055494
8   Nathaniel Wood  0.054822
9  Wanderlei Silva  0.051620
```

References

https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe

https://towardsdatascience.com/building-a-collaborative-filtering-recommender-system-with-clickstream-data-dffc86c8c65

KNN

In [47]:
```
import pandas as pd
import numpy as  np
import seaborn as sns
import os
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


os.chdir("../DataFrames")
df = pd.read_csv('finalView.csv',  index_col=0)
blueStats = pd.read_csv('blueStats.csv', index_col = 0 )
redStats = pd.read_csv('redStats.csv', index_col = 0)
blue = pd.read_csv("blue.csv")
red = pd.read_csv("red.csv")
women = pd.read_csv("womenView.csv")
```

In [48]:
```
df.head()
```

Out[48]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak	...	R_win_by_Submission	R_win_by_
0	Henry Cejudo	Marlon Moraes	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0	...	0.0	
2	Tony Ferguson	Donald Cerrone	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0	...	6.0	
3	Jimmie Rivera	Petr Yan	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	0.0	...	0.0	
4	Tai Tuivasa	Blagoy Ivanov	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	0.0	...	0.0	
6	Aljamain Sterling	Pedro Munhoz	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	False	Bantamweight	3	0.0	...	3.0	

5 rows × 146 columns

Implementing Algorithm

In [49]:
```
blueStats.drop(["B_Stance"], axis=1,inplace=True)
```

In [50]:
```
def get_recommends(fighter, stats):
    stats_rindex = stats.reset_index(drop=True)
    newView_rindex = df.reset_index(drop=True)
    dist, fighter_indices = knn.kneighbors(stats_rindex)
    fighters = fighter_indices[newView_rindex.loc[newView_rindex['B_fighter'] == fighter].index].tolist()[0]
    fighters_as_df = newView_rindex[newView_rindex.index.isin(fighters)]
    fighters_as_df = fighters_as_df[fighters_as_df['B_fighter'] != fighter]
    return fighters_as_df
```

In [51]:
```
#for the blueStats
knn = NearestNeighbors(metric = "cosine", algorithm = "brute", n_neighbors=5)
knn.fit(blueStats)
```

Out[51]:
```
NearestNeighbors(algorithm='brute', leaf_size=30, metric='cosine',
                 metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                 radius=1.0)
```

In [52]:
```
dist, fighter_indices = knn.kneighbors(blueStats)
fighter_indices[40]
```

Out[52]:
```
array([  40,  204, 1136, 1355, 1256])
```

In [53]:
```
#Find Recommendations for Amanda Nunes
```

In [54]:
```
valentinaRecommendations = get_recommends("Valentina Shevchenko", blueStats)
valentinaRecommendations
```

Out[54]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_s
618	Joseph Benavidez	Zach Makovsky	John McCarthy	2016-02-06	Las Vegas, Nevada, USA	Red	False	Flyweight	3	
630	Michael McDonald	Masanori Kanehara	Mario Yamasaki	2016-01-02	Las Vegas, Nevada, USA	Red	False	Bantamweight	3	
943	Takeya Mizugaki	Jeff Hougland	Steve Perceval	2012-11-10	Macau, China	Red	False	Bantamweight	3	
1643	Cortney Casey	Cristina Stanciu	BobbyWombacher	2016-07-13	Sioux Falls, South Dakota, USA	Red	False	Women's Strawweight	3	

4 rows × 146 columns

In [55]:
```
#Finding Recommendations for Valentina Schevchenko based on Gender
```

In [56]:
```
#shows all the Male Fighters most similar to Amanda Nunes
maleRecs = valentinaRecommendations.loc[(valentinaRecommendations['Gender'] == 1)]
maleRecs
```

Out[56]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak	...
618	Joseph Benavidez	Zach Makovsky	John McCarthy	2016-02-06	Las Vegas, Nevada, USA	Red	False	Flyweight	3	1.0	...
630	Michael McDonald	Masanori Kanehara	Mario Yamasaki	2016-01-02	Las Vegas, Nevada, USA	Red	False	Bantamweight	3	1.0	...
943	Takeya Mizugaki	Jeff Hougland	Steve Perceval	2012-11-10	Macau, China	Red	False	Bantamweight	3	1.0	...

3 rows × 146 columns

In [57]:
```
#shows all the Female Fighters more similar to Valentina Schevchenko
femaleRecs = valentinaRecommendations.loc[(valentinaRecommendations['Gender'] == 0)]
femaleRecs
```

Out[57]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_str
1643	Cortney Casey	Cristina Stanciu	BobbyWombacher	2016-07-13	Sioux Falls, South Dakota, USA	Red	False	Women's Strawweight	3	

1 rows × 146 columns

In [58]:
```
#for the redStats:
```

In [59]:
```
redStats.drop(["Referee"], axis = 1, inplace = True)
```

In [60]:
```
redStats.drop(["R_Stance"], axis =1, inplace = True)
```

In [61]:
```
knn = NearestNeighbors(metric = "cosine", algorithm = "brute", n_neighbors=5)
knn.fit(redStats)
```

Out[61]:
```
NearestNeighbors(algorithm='brute', leaf_size=30, metric='cosine',
                 metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                 radius=1.0)
```

In [62]:
```
dist, fighter_indices = knn.kneighbors(redStats)
fighter_indices[40]
```

Out[62]:
```
array([ 40, 747, 407, 122, 822])
```

In [63]:
```
valentinaRecommendations = get_recommends("Valentina Shevchenko", redStats)
valentinaRecommendations
```

In [64]:
```
#shows all the Male Fighters most similar to Valentina Schevchenko
maleRecs = valentinaRecommendations.loc[(valentinaRecommendations['Gender'] == 1)]
maleRecs
```

Out[64]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streal
419	Luke Rockhold	David Branch	Dan Miragliotta	2017-09-16	Pittsburgh, Pennsylvania, USA	Red	False	Middleweight	5	0.(
579	Luke Rockhold	Michael Bisping	John McCarthy	2016-06-04	Los Angeles, California, USA	Blue	True	Middleweight	5	0.(
1092	Brian Foster	Forrest Petz	Herb Dean	2010-09-15	Austin, Texas, USA	Red	False	Welterweight	3	2.(

3 rows × 146 columns

In [65]:
```
femaleRecs = valentinaRecommendations.loc[(valentinaRecommendations['Gender'] == 0)]
femaleRecs
```

Out[65]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak	...
1635	Amanda Nunes	Ronda Rousey	Herb Dean	2016-12-30	Las Vegas, Nevada, USA	Red	True	Women's Bantamweight	5	1.0	...

1 rows × 146 columns

Testing Accuracy

In [66]:
```
#For the blue corner
```

In [67]:
```
blueStats
```

Out[67]:

	no_of_rounds	B_current_lose_streak	B_current_win_streak	B_draw	B_avg_BODY_att	B_avg_BODY_landed	B_avg_CLINCH
0	5	0.0	4.0	0.0	9.200000	6.000000	0.20
2	3	0.0	3.0	0.0	15.354839	11.322581	6.74
3	3	0.0	4.0	0.0	17.000000	14.000000	13.75
4	3	0.0	1.0	0.0	17.000000	14.500000	2.50
6	3	0.0	3.0	0.0	15.000000	7.416667	6.08
...	...	...	...	...	...	...	
2073	3	0.0	1.0	0.0	6.333333	4.000000	4.66
2266	3	0.0	0.0	0.0	6.923077	4.823529	6.33
2352	3	0.0	0.0	0.0	6.923077	4.823529	6.33
2798	3	1.0	0.0	0.0	8.000000	6.000000	8.00
2830	3	0.0	0.0	0.0	6.923077	4.823529	6.33

1662 rows × 52 columns

In [68]:
```
df3 = blueStats
def Nearest(X_train,Y_train,X_test,Y_test,Models):
    knn = KNeighborsClassifier(n_neighbors = 5)
    knn.fit(X_train, Y_train)
    Y_pred = knn.predict(X_test)
    Models['KNN'] = [accuracy_score(Y_test,Y_pred),confusion_matrix(Y_test,Y_pred)]


def run_all_and_Plot(df3):
    Models = dict()
    from sklearn.model_selection import train_test_split
    X_all = np.asarray(df3[['B_avg_TD_pct', 'B_avg_KD', 'B_wins', 'B_avg_BODY_landed', 'B_avg_C
    y_all = np.asarray(df3['Gender'])
    X_train, X_test, Y_train, Y_test = train_test_split(X_all, y_all, test_size=0.2, random_sta
    Nearest(X_train,Y_train,X_test,Y_test,Models)
    return Models
run_all_and_Plot(df3)
```

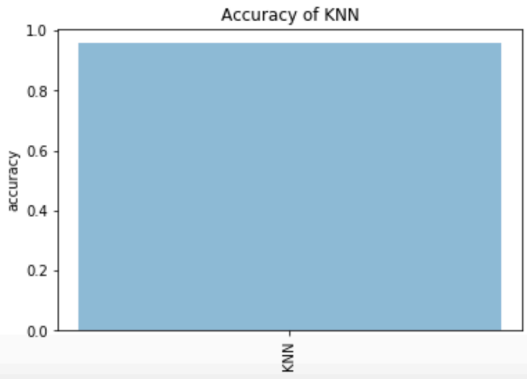
Out[68]:
```
{'KNN': [0.9579579579579579, array([[  9,  12],
         [  2, 310]])]}
```

In [69]:
```
def plot_bar(dict):
    labels = tuple(dict.keys())
    y_pos = np.arange(len(labels))
    values = [dict[n][0] for n in dict]
    plt.bar(y_pos, values, align='center', alpha=0.5)
    plt.xticks(y_pos, labels,rotation='vertical')
    plt.ylabel('accuracy')
    plt.title('Accuracy of KNN')
    plt.show()
```

In [69]:
```
def plot_bar(dict):
    labels = tuple(dict.keys())
    y_pos = np.arange(len(labels))
    values = [dict[n][0] for n in dict]
    plt.bar(y_pos, values, align='center', alpha=0.5)
    plt.xticks(y_pos, labels,rotation='vertical')
    plt.ylabel('accuracy')
    plt.title('Accuracy of KNN')
    plt.show()
```

In [70]:
```
CompareAll = dict()
accuracies = run_all_and_Plot(df3)

CompareAll['Baseline'] = accuracies
for key,val in accuracies.items():
    print(str(key) +' '+ str(val[0]))
plot_bar(accuracies)
```

```
KNN 0.9579579579579579
```



In [71]:
```
redStats
```

Out[71]:

	no_of_rounds	R_current_lose_streak	R_current_win_streak	R_draw	R_avg_CLINCH_att	R_avg_CLINCH_landed	R_avg_DISTANCE_att	R_avg_DISTANCE_lan
0	5	0.0	4.0	0.0	17.000000	11.000000	75.000000	26.500
2	3	0.0	11.0	0.0	2.866667	1.733333	116.133333	49.466
3	3	1.0	0.0	0.0	5.875000	4.125000	104.875000	41.000
4	3	1.0	0.0	0.0	11.000000	7.250000	50.750000	24.750
6	3	0.0	3.0	0.0	14.000000	11.083333	72.583333	29.416
...	...	...	...	...	...	...	...	
2073	3	0.0	2.0	0.0	15.250000	10.000000	44.000000	16.500
2266	3	0.0	2.0	0.0	21.666667	13.666667	148.666667	79.333
2352	3	0.0	0.0	0.0	6.500000	4.300000	45.416667	15.750
2798	3	0.0	0.0	0.0	6.500000	4.300000	45.416667	15.750
2830	3	0.0	0.0	0.0	6.500000	4.300000	45.416667	15.750

1662 rows × 63 columns

In [72]:
```
#For the red corner
```

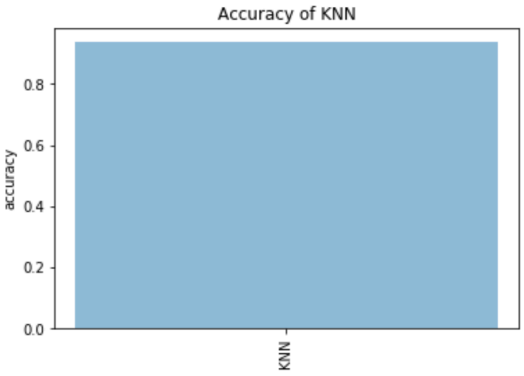
In [73]:
```
df3 = redStats


def Nearest(X_train,Y_train,X_test,Y_test,Models):
    knn = KNeighborsClassifier(n_neighbors = 5)
    knn.fit(X_train, Y_train)
    Y_pred = knn.predict(X_test)
    Models['KNN'] = [accuracy_score(Y_test,Y_pred),confusion_matrix(Y_test,Y_pred)]


def run_all_and_Plot(df3):
    Models = dict()
    from sklearn.model_selection import train_test_split
    X_all = np.asarray(df3[['R_avg_TD_pct', 'R_avg_KD', 'R_wins', 'R_avg_CLINCH_landed', 'R_avg
    y_all = np.asarray(df3['Gender'])
    X_train, X_test, Y_train, Y_test = train_test_split(X_all, y_all, test_size=0.2, random_sta
    Nearest(X_train,Y_train,X_test,Y_test,Models)
    return Models
run_all_and_Plot(df3)
```

Out[73]:
```
{'KNN': [0.9369369369369369, array([[  3,  18],
        [  3, 309]])]}
```

In [74]:
```
def plot_bar(dict):
    labels = tuple(dict.keys())
    y_pos = np.arange(len(labels))
    values = [dict[n][0] for n in dict]
    plt.bar(y_pos, values, align='center', alpha=0.5)
    plt.xticks(y_pos, labels,rotation='vertical')
    plt.ylabel('accuracy')
    plt.title('Accuracy of KNN')
    plt.show()
```

In [75]:
```
CompareAll = dict()
accuracies = run_all_and_Plot(df3)

CompareAll['Baseline'] = accuracies
for key,val in accuracies.items():
    print(str(key) +' '+ str(val[0]))
plot_bar(accuracies)
```

```
KNN 0.9369369369369369
```



Loading the Data

In [1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import datasets, linear_model
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import mean_squared_error, r2_score
os.chdir("../DataFrames")
finalView = pd.read_csv('finalView.csv',  index_col=0)
redStats = pd.read_csv("redStats.csv", index_col = 0)
blueStats = pd.read_csv("blueStats.csv", index_col =0)
red = pd.read_csv("red.csv", index_col =0)
```

In [2]:
```
finalView.head()
```

Out[2]:

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak	...	R_win_by_Submission	R_win_by_
0	Henry Cejudo	Marlon Moraes	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0	...	0.0	
2	Tony Ferguson	Donald Cerrone	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0	...	6.0	
3	Jimmie Rivera	Petr Yan	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	0.0	...	0.0	
4	Tai Tuivasa	Blagoy Ivanov	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	0.0	...	0.0	
6	Aljamain Sterling	Pedro Munhoz	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	False	Bantamweight	3	0.0	...	3.0	

5 rows × 146 columns

Linear Regression

Training against the data, the prediction in this data set's case would be the winner's win-streak. Splitting the dataset between the blue winner and the red winner to see their stats would be beneficial to see how opposing sides perform (and whether or not the corner assigned details any relevant information). Initially, it looks at just the win streak for the fighter and their ability to knockout an opponent. This gives us a comparison to initially see. Then, the statistics, to optimize run time, will be displayed as averages overall.

In [3]:
```
red
```

Out[3]:

	R_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	R_current_lose_streak	R_current_win_streak	...	R_win_by_KO/TK
0	Henry Cejudo	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0	4.0	...	2.
2	Tony Ferguson	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0	11.0	...	3.
3	Jimmie Rivera	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	1.0	0.0	...	1.
4	Tai Tuivasa	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	1.0	0.0	...	2.
6	Aljamain Sterling	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	False	Bantamweight	3	0.0	3.0	...	1.
...	...	...	...	...	...	...	...	...	...	...	...	.
2073	Miesha Tate	Marc Goddard	2015-01-31	Las Vegas, Nevada, USA	Red	False	Women's Bantamweight	3	0.0	2.0	...	0.
2266	Jessica Andrade	Fernando Yamasaki	2014-09-13	Brasilia, Distrito Federal, Brazil	Red	False	Women's Bantamweight	3	0.0	2.0	...	0.
2352	Claudia Gadelha	Liam Kerrigan	2014-07-16	Atlantic City, New Jersey, USA	Red	False	Women's Strawweight	3	0.0	0.0	...	0.
2798	Amanda Nunes	Herb Dean	2013-08-03	Rio de Janeiro,	Red	False	Women's	3	0.0	0.0	...	0.

2798	Amanda Nunes	Herb Dean	2013-08-03	Rio de Janeiro, Brazil	Red	False	Women's Bantamweight	3	0.0
2830	Alexis Davis	Herb Dean	2013-06-15	Winnipeg, Manitoba, Canada	Red	False	Women's Bantamweight	3	0.0

1662 rows × 71 columns

In [5]:
```
red.head()
```

Out[5]:

	R_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	R_current_lose_streak	R_current_win_
0	Henry Cejudo	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0.0	
2	Tony Ferguson	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0.0	
3	Jimmie Rivera	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	1.0	
4	Tai Tuivasa	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	1.0	
6	Aljamain Sterling	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	False	Bantamweight	3	0.0	

5 rows × 71 columns

In [6]:
```
red.drop(["Referee", "date", "location", "Winner", "title_bout", "weight_class"], axis = 1, inp
```

In [7]:
```
red
```

Out[7]:

	R_fighter	no_of_rounds	R_current_lose_streak	R_current_win_streak	R_draw	R_avg_CLINCH_att	R_avg_CLINCH_landed
0	Henry Cejudo	5	0.0	4.0	0.0	17.000000	11.000000
2	Tony Ferguson	3	0.0	11.0	0.0	2.866667	1.733333
3	Jimmie Rivera	3	1.0	0.0	0.0	5.875000	4.125000
4	Tai Tuivasa	3	1.0	0.0	0.0	11.000000	7.250000
6	Aljamain Sterling	3	0.0	3.0	0.0	14.000000	11.083333
...	...	...	...	...	...	...	...
2073	Miesha Tate	3	0.0	2.0	0.0	15.250000	10.000000
2266	Jessica Andrade	3	0.0	2.0	0.0	21.666667	13.666667
2352	Claudia Gadelha	3	0.0	0.0	0.0	6.500000	4.300000
2798	Amanda Nunes	3	0.0	0.0	0.0	6.500000	4.300000
2830	Alexis Davis	3	0.0	0.0	0.0	6.500000	4.300000

1662 rows × 65 columns

Just looking at TKOs

In [7]:
```
#For the red fighters.
```

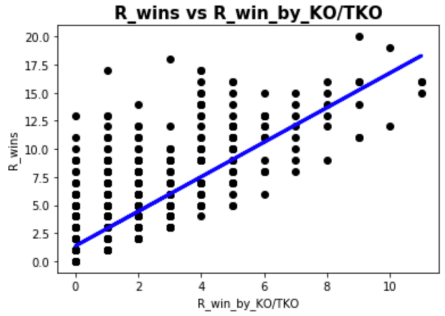
In [8]:
```
def split_fighter_datasets(predictor, target,viewName):

    #Split arrays or matrices into random train and test subsets
    X = viewName[predictor]
    y = viewName[target]
    fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test = train_test_split(X, y, t
    return fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test
```

Training the data

In [9]:
```
predictor=['R_win_by_KO/TKO']
target='R_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,
regr = linear_model.LinearRegression()
```

In [10]:
```
def calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test
    # Train the model using the training sets
    regr.fit(fighter_X_train,fighter_y_train)
    # The coefficients
    print('Coefficients: \n', regr.coef_)
    # The mean squared error
    print('Mean squared error: %.2f'
          % mean_squared_error(fighter_y_test, regr.predict(fighter_X_test)))
    # The coefficient of determination: 1 is perfect prediction
    print('Coefficient of determination: %.2f' % r2_score(fighter_y_test, regr.predict(fighter_X_test)))

    print ('Variance Score: %.2f' % regr.score(fighter_X_test, fighter_y_test))

    # Plot outputs
    plt.scatter(fighter_X_test, fighter_y_test,  color='black')
    plt.plot(fighter_X_test, regr.predict(fighter_X_test), color='blue', linewidth=3)
    plt.xlabel(predictor[0])
    plt.ylabel(target)
    plt.title(target + " vs " + predictor[0], fontsize=15, fontweight='bold')

    plt.show()
calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test)
```

```
Coefficients:
 [1.54095507]
Mean squared error: 6.72
Coefficient of determination: 0.56
Variance Score: 0.56
```



In [11]:
```
#looking at the Blue fighters TKO's.
blueStats
```

Out[11]:

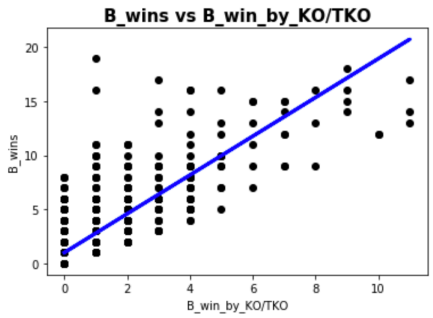
	no_of_rounds	B_current_lose_streak	B_current_win_streak	B_draw	B_avg_BODY_att	B_avg_BODY_landed	B_avg_CLINCH_att	B_avg_CLINCH_landed	B_a
0	5	0.0	4.0	0.0	9.200000	6.000000	0.200000	0.000000	
2	3	0.0	3.0	0.0	15.354839	11.322581	6.741935	4.387097	
3	3	0.0	4.0	0.0	17.000000	14.000000	13.750000	11.000000	
4	3	0.0	1.0	0.0	17.000000	14.500000	2.500000	2.000000	
6	3	0.0	3.0	0.0	15.000000	7.416667	6.083333	3.416667	
...	...	...	...	...	...	...	...	...	
2073	3	0.0	1.0	0.0	6.333333	4.000000	4.666667	3.666667	
2266	3	0.0	0.0	0.0	6.923077	4.823529	6.333333	4.200000	
2352	3	0.0	0.0	0.0	6.923077	4.823529	6.333333	4.200000	
2798	3	1.0	0.0	0.0	8.000000	6.000000	8.000000	6.000000	
2830	3	0.0	0.0	0.0	6.923077	4.823529	6.333333	4.200000	

1662 rows × 53 columns

In [12]:
```
predictor=['B_win_by_KO/TKO']
target='B_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,target,blueStats)
```

In [13]:
```
def calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test):
  # Train the model using the training sets
  regr.fit(fighter_X_train,fighter_y_train)
  # The coefficients
  print('Coefficients: \n', regr.coef_)
  # The mean squared error
  print('Mean squared error: %.2f'
        % mean_squared_error(fighter_y_test, regr.predict(fighter_X_test)))
  # The coefficient of determination: 1 is perfect prediction
  print('Coefficient of determination: %.2f' % r2_score(fighter_y_test, regr.predict(fighter_X_test)))

  print ('Variance Score: %.2f' % regr.score(fighter_X_test, fighter_y_test))

  # Plot outputs
  plt.scatter(fighter_X_test, fighter_y_test,  color='black')
  plt.plot(fighter_X_test, regr.predict(fighter_X_test), color='blue', linewidth=3)
  plt.xlabel(predictor[0])
  plt.ylabel(target)
  plt.title(target + " vs " + predictor[0], fontsize=15, fontweight='bold')

  plt.show()
alculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test)
```

```
Coefficients:
 [1.79382125]
Mean squared error: 4.27
Coefficient of determination: 0.61
Variance Score: 0.61
```



Computing Averages...

In [14]:
```
def pairwise_metrics(viewColor, stringTarget):
#    Updates a dictionary with its metrics for  the pairwise features in the dataset.
#    And to produce a pair of features most similar to each other
#    Which would be beneficial in determining fighter's similarity.
#     Args:
#       viewColor, the view you wish to use
#       stringTarget the target value you wish to use


    i = 1

    # Divide columns into features and class.
    features = list(viewColor.columns)
    #classes = features[-1] # create class column
    #del features[-1] # delete class column from feature vector
    colorsDictionary = {"Coefficients": None, "Mean Squared Error": None, "Coefficient of determination": None, "Varian
    listCoef = []
    listMSE = []
    listCoefOfDetermination = []
    listVariance = []

    # Generate an nxn subplot figure, where n is the number of features.
#     figure = pl.figure(figsize=(5*(len(viewColor.columns)-1), 4*(len(viewColor.columns)-1)))
    for col1 in viewColor[features]:
        for col2 in viewColor[features]:
                target = str(stringTarget)
                #Split arrays or matrices into random train and test subsets
                fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test = split_fighter_datasets(col1, target,

                #Gives a new shape to an array without changing its data.
                fighter_X_train = fighter_X_train.values.reshape(-1, 1)
                fighter_y_train = fighter_y_train.values.reshape(-1, 1)
                fighter_X_test = fighter_X_test.values.reshape(-1, 1)
                fighter_y_test = fighter_y_test.values.reshape(-1, 1)
                # Create linear regression object
                reg = linear_model.LinearRegression()
                # Train the model using the training sets

                reg.fit(fighter_X_train,fighter_y_train)
                # The coefficients

                listCoef.append(reg.coef_)
```

```
                # The mean squared error

                listMSE.append(mean_squared_error(fighter_y_test, reg.predict(fighter_X_test)))

                # The coefficient of determination: 1 is perfect prediction
                listCoefOfDetermination.append( r2_score(fighter_y_test, reg.predict(fighter_X_

                #The variance score
                listVariance.append(reg.score(fighter_X_test, fighter_y_test))

    return listCoef,listMSE, listCoefOfDetermination, listVariance
```

In [15]:
```
def listAverages (listCoef, listMSE, listCoefOfDetermination, listVariance):
    dictionary = {"Coefficients": None, "Mean Squared Error": None, "Coefficient of determinati
    counter1 = 0
    sum1 = 0
    sum2 = 0
    sum3 = 0
    sum4 = 0
    for i in listCoef:
        sum1+=i
        counter1+=1
    for j in listMSE:
        sum2+=j
    for k in listCoefOfDetermination:
        sum3+=i
    for alpha in listVariance:
        sum4+=alpha

    dictionary.update({"Coefficients": sum1/counter1})
    dictionary.update({"Mean Squared Error": sum2/counter1})
    dictionary.update({"Coefficient of determination": sum3/counter1})
    dictionary.update({"Variance Score": sum4/counter1})


    return dictionary
```

In [16]:
```
#For the Blue Fighters
```

In [18]:
```
blueStats.drop(["B_Stance"], axis=1,inplace=True)
blueCoef, blueMSE, blueCoefOfDetermination, blueVariance = pairwise_metrics(blueStats, "B_wins" )
```

In [19]:
```
blueAverages = listAverages(blueCoef, blueMSE, blueCoefOfDetermination, blueVariance)
```

In [20]:
```
blueAverages
```

Out[20]:
```
{'Coefficients': array([[0.71241854]]),
 'Mean Squared Error': 9.808457928117164,
 'Coefficient of determination': array([[0.55570535]]),
 'Variance Score': 0.11612557950513307}
```

In [21]:
```
blueMetrics = pd.DataFrame([blueAverages], columns=blueAverages.keys())
```

In [22]:
```
blueMetrics
```

Out[22]:

	Coefficients	Mean Squared Error	Coefficient of determination	Variance Score
0	[[0.7124185426105766]]	9.808458	[[0.5557053514586049]]	0.116126

In [23]:
```
#For the Red Fighters
```

In [24]:
```
redStats.drop(["Referee"], axis =1, inplace=True)
```

In [25]:
```
redStats.drop(["R_Stance"], axis =1, inplace=True)
```

In [26]:
```
redCoef, redMSE, redCoefOfDetermination, redVariance = pairwise_metrics(redStats, "R_wins")
```

In [27]:
```
redAverages = listAverages(redCoef, redMSE, redCoefOfDetermination, redVariance)
```

In [28]:
```
redMetrics = pd.DataFrame([redAverages], columns=redAverages.keys())
```

In [29]:
```
redMetrics
```

Out[29]:

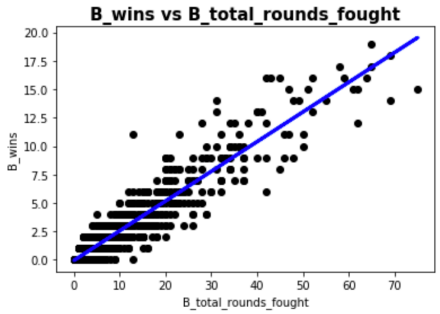
	Coefficients	Mean Squared Error	Coefficient of determination	Variance Score
0	[[0.44694159048449505]]	13.972284	[[0.8642849118077383]]	0.089855

Determining the best fit pair of features

In [30]:
```
predictor=['B_total_rounds_fought']
target='B_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,target,blueStats)
```

In [31]:
```
calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test)
```

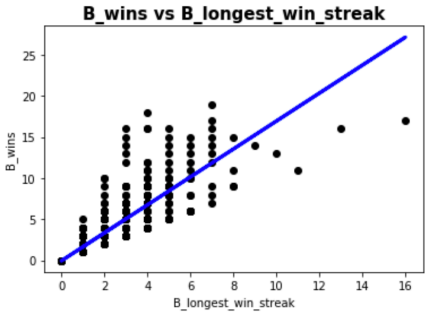
```
Coefficients:
 [0.26128101]
Mean squared error: 1.36
Coefficient of determination: 0.88
Variance Score: 0.88
```



In [32]:
```
predictor=['B_longest_win_streak']
target='B_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,target,blueStats)
```

In [33]:
```
calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test)
```

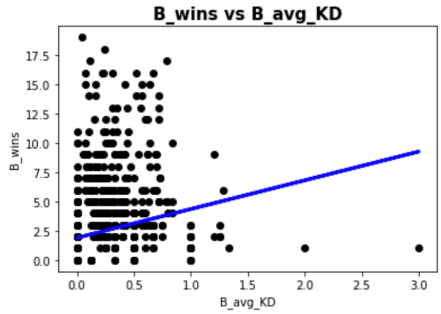
```
Coefficients:
 [1.70022375]
Mean squared error: 2.77
Coefficient of determination: 0.75
Variance Score: 0.75
```



In [34]:
```
predictor=['B_avg_KD']
target='B_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,target,blueStats)
```

In [35]:
```
calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test)
```

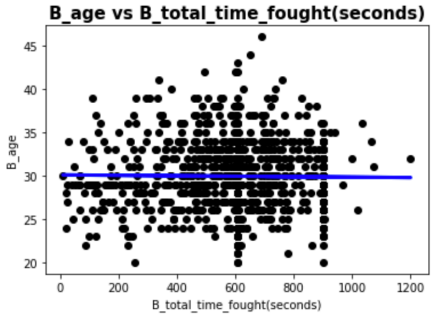
```
Coefficients:
 [2.45395346]
Mean squared error: 10.36
Coefficient of determination: 0.07
Variance Score: 0.07
```



In [36]:
```
predictor=['B_total_time_fought(seconds)']
target='B_age'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,target,blueStats)
```

In [37]:
```
calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_train, fighter_y_test)
```

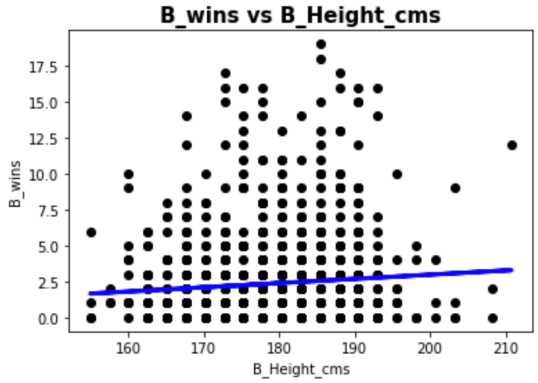
```
Coefficients:
 [-0.00025583]
Mean squared error: 15.77
Coefficient of determination: -0.00
Variance Score: -0.00
```



In [38]:
```
predictor=['B_Height_cms']
target='B_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,target,blueStats)
```

In [39]:
```
calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_t
```

```
Coefficients:
 [0.02932905]
Mean squared error: 11.10
Coefficient of determination: -0.00
Variance Score: -0.00
```

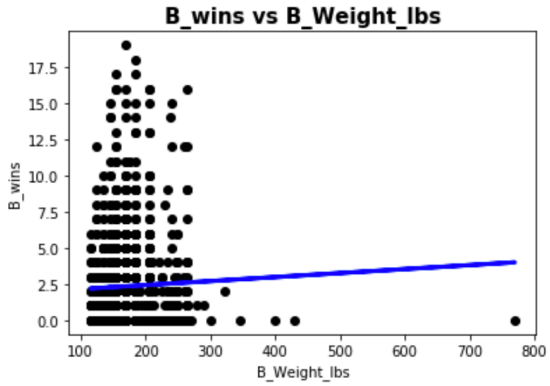


In [40]:
```
predictor=['B_Weight_lbs']
target='B_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,
```

In [40]:
```
predictor=['B_Weight_lbs']
target='B_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,
```

In [41]:
```
calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_t
```

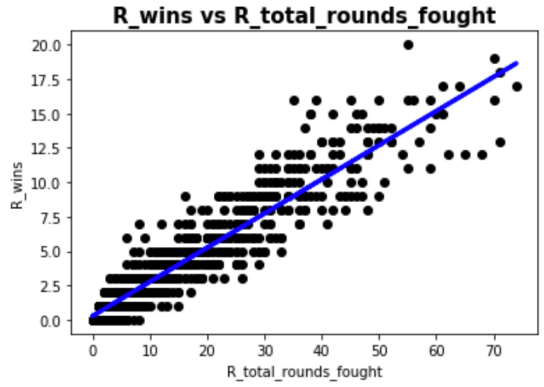
```
Coefficients:
 [0.00276127]
Mean squared error: 11.14
Coefficient of determination: -0.00
Variance Score: -0.00
```



In [42]:
```
redStats
predictor=['R_total_rounds_fought']
target='R_wins'
fighter_X_train,fighter_X_test,fighter_y_train,fighter_y_test=split_fighter_datasets(predictor,
```

In [43]:
```
calculate_and_plot_fighter_linear_regression(regr, fighter_X_train, fighter_X_test, fighter_y_t
```

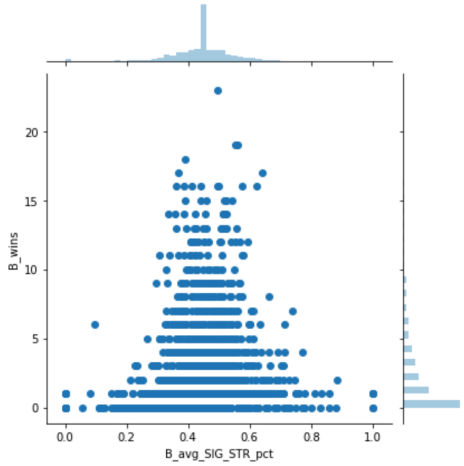
```
Coefficients:
 [0.24839598]
Mean squared error: 1.76
Coefficient of determination: 0.89
Variance Score: 0.89
```



Lineplots

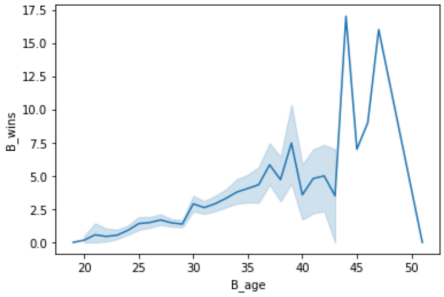
In [2]:
```
#For the Blue Corner
```

In [3]:
```
df2 = pd.read_csv("finalView.csv")
sns.jointplot(x="B_avg_SIG_STR_pct", y="B_wins", data=df2);
```



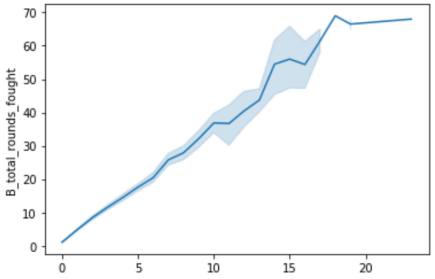
In [4]:
```
#A line plot for B_age and B_wins
sns.lineplot(x="B_age", y="B_wins", data=df2)
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b6958d0>



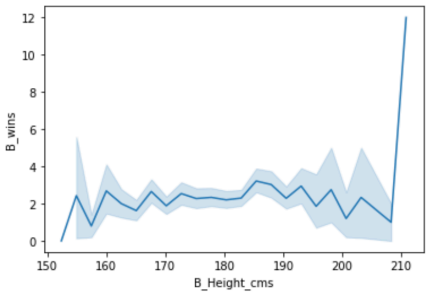
In [5]:
```
sns.lineplot(x="B_wins", y = "B_total_rounds_fought", data=df2)
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b78fd50>



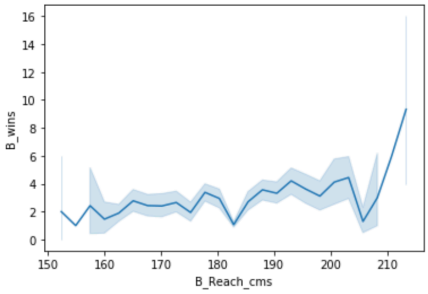
In [6]:
```
#A line plot between B height and B Wins
sns.lineplot(x="B_Height_cms", y="B_wins", data=df2)
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b87a450>



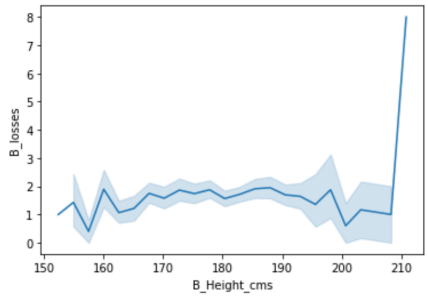
In [7]:
```
#A line plot between B Reach and B Wins
sns.lineplot(x="B_Reach_cms", y="B_wins", data=df2)
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1b966390>



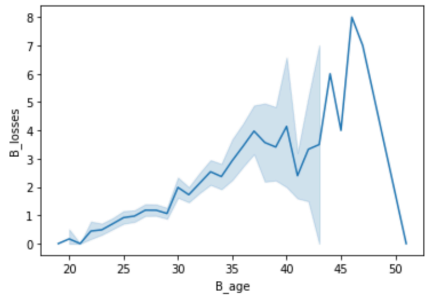
In [8]:
```
#A line plot between B Height and B Losses
sns.lineplot(x="B_Height_cms", y="B_losses", data=df2)
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1ba51710>



In [9]:
```
#A line plot betwen B and and B losses
sns.lineplot(x="B_age", y="B_losses", data=df2)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1bb403d0>

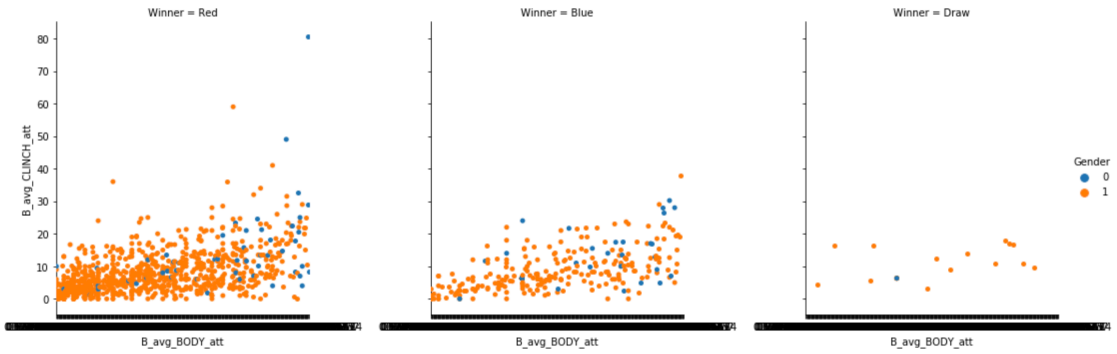


Categorical Plots

In [12]:
```
#Based on gender.
```

In [13]:
```
#A categorical plot separating out gender, their average body attacks and average clinch attacks
import seaborn as sns
df2 = finalView
sns.catplot(x="B_avg_BODY_att", y="B_avg_CLINCH_att", hue="Gender", col="Winner",  data=df2)
```

Out[13]: <seaborn.axisgrid.FacetGrid at 0x1a1b68a990>



In [20]:
```
import seaborn as sns
sns.catplot(x="B_avg_BODY_landed", y="B_avg_CLINCH_landed", hue="Gender", col="Winner", data=df
sns.catplot(x="B_avg_HEAD_landed", y="B_avg_GROUND_landed", hue="Gender", col="Winner", data=df
```

Out[20]:
```
<seaborn.axisgrid.FacetGrid at 0x1a1d436a90>
```