FanduelAssessment

April 10, 2024

[1]:
```
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
```

[2]:
```
df = pd.read_csv('ProjectDataset.csv')
```

[3]:
```
df.head()
```

[3]:
```
    state      playerid       wagerid                event_start placed_date  \
0  State1  3.065121e+07  1.693004e+06  2021-04-28 00:30:00+00  2021-04-27
1  State1  2.223717e+07  1.696371e+06  2021-04-28 01:45:00+00  2021-04-27
2  State1  2.223717e+07  1.696371e+06  2021-04-28 01:45:00+00  2021-04-27
3  State1  2.223717e+07  1.696371e+06  2021-04-28 01:45:00+00  2021-04-27
4  State1  2.223717e+07  1.696371e+06  2021-04-28 01:45:00+00  2021-04-27

  settled_date sportname  bet_type result  net_stake   ggr legresult  \
0   2021-04-27       nhl  straight    won       6.64 -4.96       won
1   2021-04-27       nba    parlay   lost       5.00  5.00       won
2   2021-04-27       nba    parlay   lost       5.00  5.00      lost
3   2021-04-27       nba    parlay   lost       5.00  5.00      lost
4   2021-04-27       nba    parlay   lost       5.00  5.00      lost

   decimalodds
0      1.74627
1      1.78125
2      1.86207
3      1.74627
4      1.78125
```

[4]:
```
df['placed_date'] = pd.to_datetime(df['placed_date'])
df['settled_date'] = pd.to_datetime(df['settled_date'])
```

[5]:
```
# Check for null values
print(df.isnull().sum())
```

```
state              0
playerid           0
wagerid            0
event_start        0
placed_date        0
settled_date       0
sportname          0
bet_type           0
result             0
net_stake          0
ggr                0
legresult          0
decimalodds     2472
dtype: int64
```

[6]:
```
# Since we have null values in our 'decimalodds' column, I will handle these in␣
↪two different ways
```

[7]:
```
# Method 1: Predict null values
data_imputed = df
imputer = SimpleImputer(strategy='mean')
data_imputed['decimalodds'] = imputer.
↪fit_transform(data_imputed[['decimalodds']])
#print(data_imputed.isnull().sum())
```

[8]:
```
# Method 2: remove rows with null values
data_clean = df
data_clean.dropna(subset=['decimalodds'], inplace=True)
#print(data_clean.isnull().sum())
```

[9]:
```
#data_clean['event_start']
```

[10]:
```
data_clean['event_start_date'] = data_clean['event_start'].str[:10]
data_clean['event_start_date'] = pd.to_datetime(data_clean['event_start_date'],␣
↪errors='coerce')
```

[11]:
```
'''
Ideas:
1) How does distance between placed_date and event_start relate to net_stake.
    Are user placing larger wagers closer to event_start or further from start␣
↪(closer to market opening)?
    Filter for place_date to be max single week away from settle_date to avoid␣
↪futures.

2) GGR vs distance between placed_date and event_start. Do we make more money␣
↪on bets placed closer to event_start?
    Filter for futures.
```

```
    Actionable insight: Open "bet boosts" or raise limits on boosts closer to␣
 ↪event_start.

3) Check how profitable future bets are for the book. Filter for bets placed 2+␣
 ↪weeks before settle date.
    What % of users are placing future bets.
    Actionable insight: If % is low and futures are profitable, push marketing␣
 ↪of future bets.
    Season-long futures are often marketed close to season open what if we have␣
 ↪week-long mid-season promotion on championship futures only.

4) Get playerids in 98 percentile of wagers place, total wagered, GGR to␣
 ↪identify power users.
    Actionable insight: Bonuses to top GGR players (biggest losers)

5) Check correlation between sportname and GGR. Can filter for only straight␣
 ↪bets as parlays likely strongly correlate with high GGR.

6)
'''
```

[11]:

```
'\nIdeas:\n1) How does distance between placed_date and event_start relate to
net_stake.\n    Are user placing larger wagers closer to event_start or further
from start (closer to market opening)?\n    Filter for place_date to be max
single week away from settle_date to avoid futures.\n    \n2) GGR vs distance
between placed_date and event_start. Do we make more money on bets placed closer
to event_start?\n    Filter for futures.\n    Actionable insight: Open "bet
boosts" or raise limits on boosts closer to event_start.\n    \n3) Check how
profitable future bets are for the book. Filter for bets placed 2+ weeks before
settle date.\n    What % of users are placing future bets. \n    Actionable
insight: If % is low and futures are profitable, push marketing of future
bets.\n    Season-long futures are often marketed close to season open what if
we have week-long mid-season promotion on championship futures only.\n    \n4)
Get playerids in 98 percentile of wagers place, total wagered, GGR to identify
power users. \n    Actionable insight: Bonuses to top GGR players (biggest
losers)\n    \n5) Check correlation between sportname and GGR. Can filter for
only straight bets as parlays likely strongly correlate with high GGR.\n\n6) \n'
```

[12]:

```
# Exploratory Analysis
data = data_clean
data.describe()
# doesn't offer any insight in our case
```

[12]:

	playerid	wagerid	net_stake	ggr	decimalodds
count	4.174500e+06	4.174500e+06	4.174500e+06	4.174500e+06	4.174500e+06
mean	2.273927e+07	4.754173e+07	2.361655e+01	2.451643e+00	3.984459e+00
std	1.033895e+07	3.675171e+07	8.189246e+01	8.532259e+01	1.602455e+01

```
min    5.971640e+04  1.691622e+06  4.001000e+00 -2.100000e+04  1.000100e+00
25%    1.750210e+07  2.296253e+07  5.000000e+00  0.000000e+00  1.649350e+00
50%    2.379874e+07  3.859404e+07  1.000000e+01  6.000000e+00  1.909090e+00
75%    3.064089e+07  5.861577e+07  2.000000e+01  1.200000e+01  2.200000e+00
max    4.297671e+07  1.779192e+08  1.967200e+04  1.967200e+04  5.001000e+03
```

[13]:

```
# Over what time period is the data?
start = data['settled_date'].min()
end = data['settled_date'].max()
span = end - start
print(start, end)
```

```
2021-03-28 00:00:00 2022-03-29 00:00:00
```

[14]:

```
# How many states and how many users are included
data.nunique()
```

[14]:

```
state                   3
playerid            36387
wagerid           2402300
event_start         11991
placed_date           366
settled_date          367
sportname               7
bet_type                2
result                  2
net_stake           31701
ggr                 96979
legresult               5
decimalodds         18695
event_start_date      385
dtype: int64
```

[15]:

```
# What are the most popular sports to bet on?
print(data['sportname'].value_counts())
```

```
nfl                    1373791
nba                    1359421
college basketball      496810
mlb                     475550
college football        260890
nhl                     187644
champions league         20394
Name: sportname, dtype: int64
```

[16]:

```
# What are the most popular sports to bet straight on vs parlay?
straight_bets = data[data['bet_type'] == 'straight']
straight_sports_counts = straight_bets['sportname'].value_counts()
```

```
print("Most bet on sports (straight bets):\n",straight_sports_counts)

parlay_bets = data[data['bet_type'] == 'parlay']
parlay_sports_counts = parlay_bets['sportname'].value_counts()

print("\nMost bet on sports (parlay bets):\n",parlay_sports_counts)
```

```
Most bet on sports (straight bets):
 nfl                    537401
nba                    506114
college basketball     250995
mlb                    200021
college football       130282
nhl                     82590
champions league         7499
Name: sportname, dtype: int64

Most bet on sports (parlay bets):
 nba                    853307
nfl                    836390
mlb                    275529
college basketball     245815
college football       130608
nhl                    105054
champions league        12895
Name: sportname, dtype: int64
```

[17]:
```
# Get column for distance between place and settle
data['betdistance'] = data['settled_date'] - data['placed_date']
# Create column to identify bets likley to be futures
data['isfuture'] = np.where(data['betdistance'] >= pd.to_timedelta(14,␣
 ↪unit='D'), True, False)
print(data['isfuture'].value_counts())
```

```
False    4156140
True       18360
Name: isfuture, dtype: int64
```

[18]:
```
# Separate futures and non-futures into different dataframes
f_df = data[data['isfuture'] == True]
cur_df = data[data['isfuture'] == False]
```

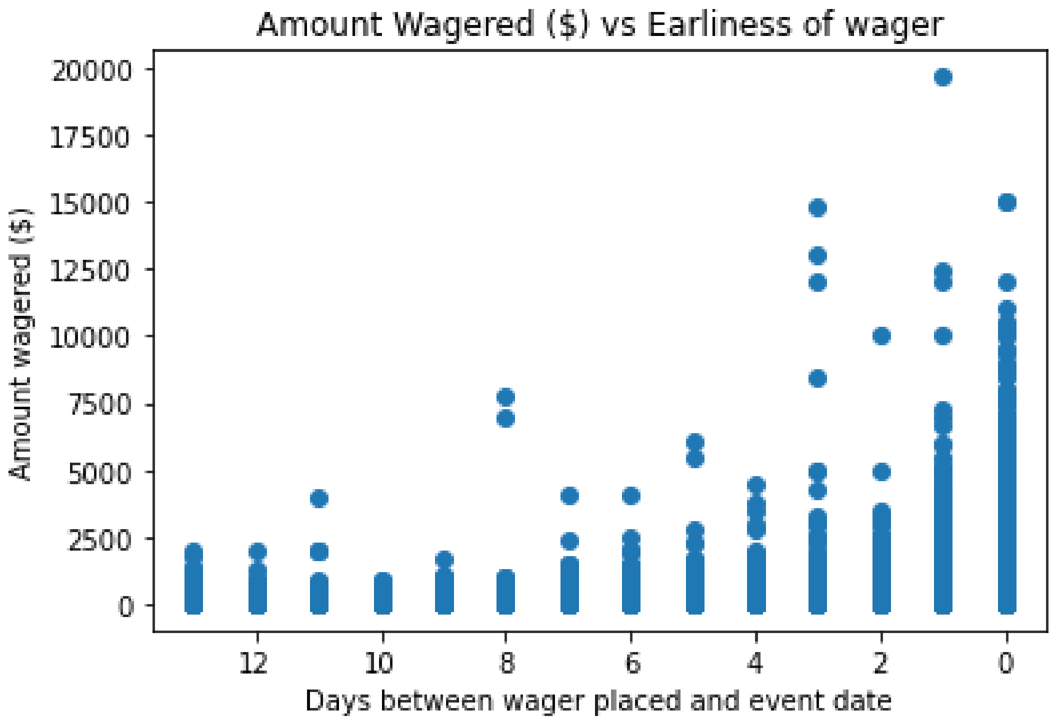
[19]:
```
'''1) How does distance between placed_date and event_start relate to net_stake.
    Are user placing larger wagers closer to event_start or further from start␣
 ↪(closer to market opening)?
    Filter for place_date to be max single week away from settle_date to avoid␣
 ↪futures.'''
```

[19]: '1) How does distance between placed_date and event_start relate to net_stake.\n
Are user placing larger wagers closer to event_start or further from start
(closer to market opening)?\n   Filter for place_date to be max single week
away from settle_date to avoid futures.'

[20]:
```
# Using cur_df, the data frame where I've removed futures, lets check bet␣
↪distance vs net stake
cur_df['betdistance'] = cur_df['betdistance'].apply(lambda x: int(x.days))
```

```
<ipython-input-20-10849b7399fa>: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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  cur_df['betdistance'] = cur_df['betdistance'].apply(lambda x: int(x.days))
```

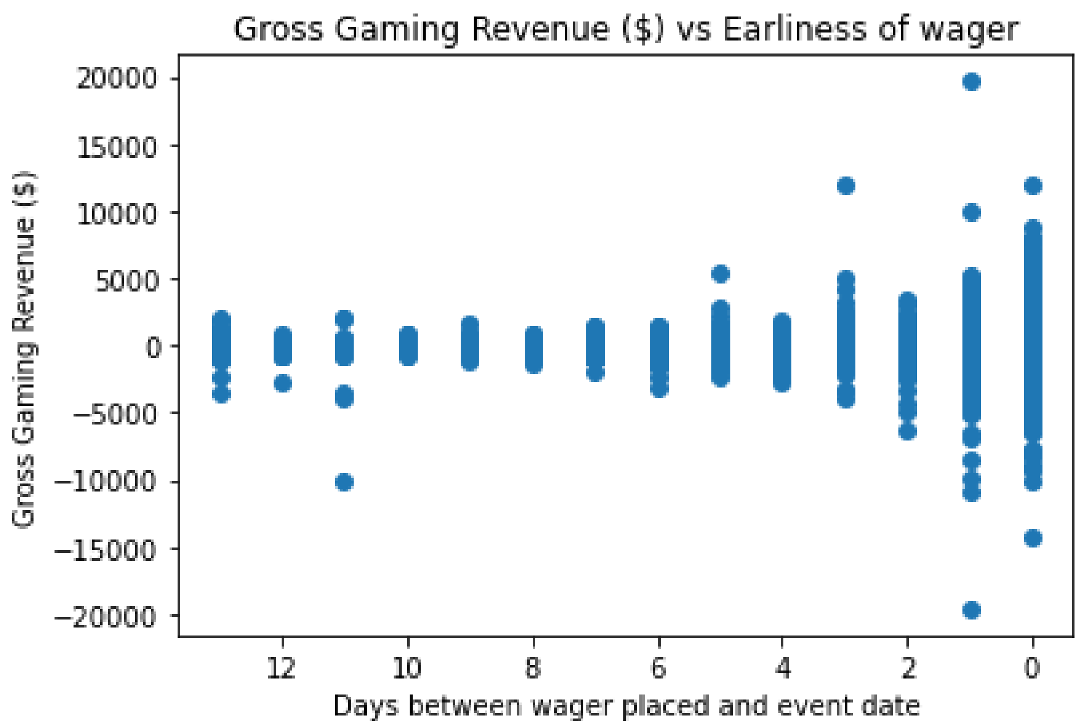
[21]:
```
fig, ax = plt.subplots()
ax.invert_xaxis()
ax.set_xlabel("Days between wager placed and event date")
ax.set_ylabel("Amount wagered ($)")
ax.set_title("Amount Wagered ($) vs Earliness of wager")
#ax.plot(cur_df['betdistance'], trendline(cur_df['betdistance']), color='red')
ax.scatter(x=cur_df['betdistance'], y=cur_df['net_stake'])



plt.show()
```



[22]:
```
'''2) GGR vs distance between placed_date and event_start. Do we make more␣
↪money on bets placed closer to event_start?
    Filter for futures.
    Actionable insight: Open "bet boosts" or raise limits on boosts closer to␣
↪event_start.'''
```

[22]:
```
'2) GGR vs distance between placed_date and event_start. Do we make more money
on bets placed closer to event_start?\n    Filter for futures.\n    Actionable
insight: Open "bet boosts" or raise limits on boosts closer to event_start.'
```

[23]:
```
fig, ax = plt.subplots()
ax.invert_xaxis()
ax.set_xlabel("Days between wager placed and event date")
ax.set_title("Gross Gaming Revenue ($) vs Earliness of wager")
ax.set_ylabel("Gross Gaming Revenue ($)")
#ax.plot(cur_df['betdistance'], trendline(cur_df['betdistance']), color='red')
ax.scatter(x=cur_df['betdistance'], y=cur_df['ggr'])


plt.show()
```



[24]:
```
# insignificant, as we get closer to event start wagers grow which explains the␣
↪growing spread
# in ggr when looking at individual bets
# both winning and losing wagers are larger
```

[25]:
```
sum_ggr_by_betdistance = cur_df.groupby('betdistance')['ggr'].sum()
sum_ggr_by_betdistance
```

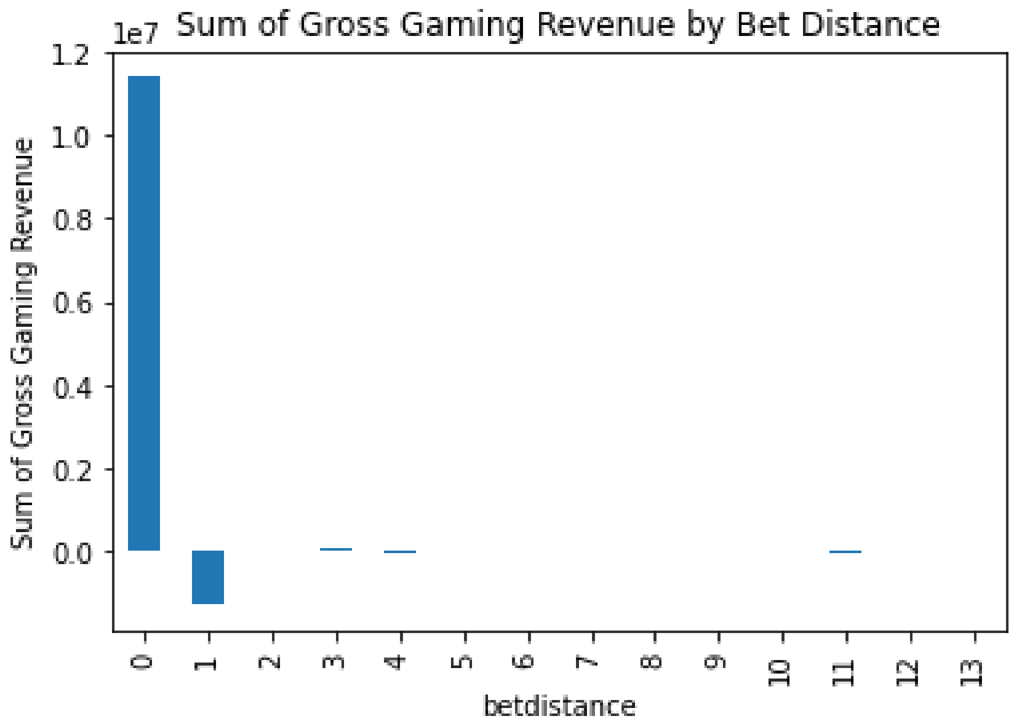
[25]:
```
betdistance
0     1.139105e+07
1    -1.262747e+06
2    -5.034554e+02
3     9.531772e+04
4    -1.773067e+04
5     3.378697e+04
6     1.481271e+04
7     3.842742e+02
8     7.163955e+03
9     1.150763e+04
10    1.082258e+04
11   -9.187369e+03
12    1.490307e+02
13    7.144369e+03
```

```
Name: ggr, dtype: float64
```

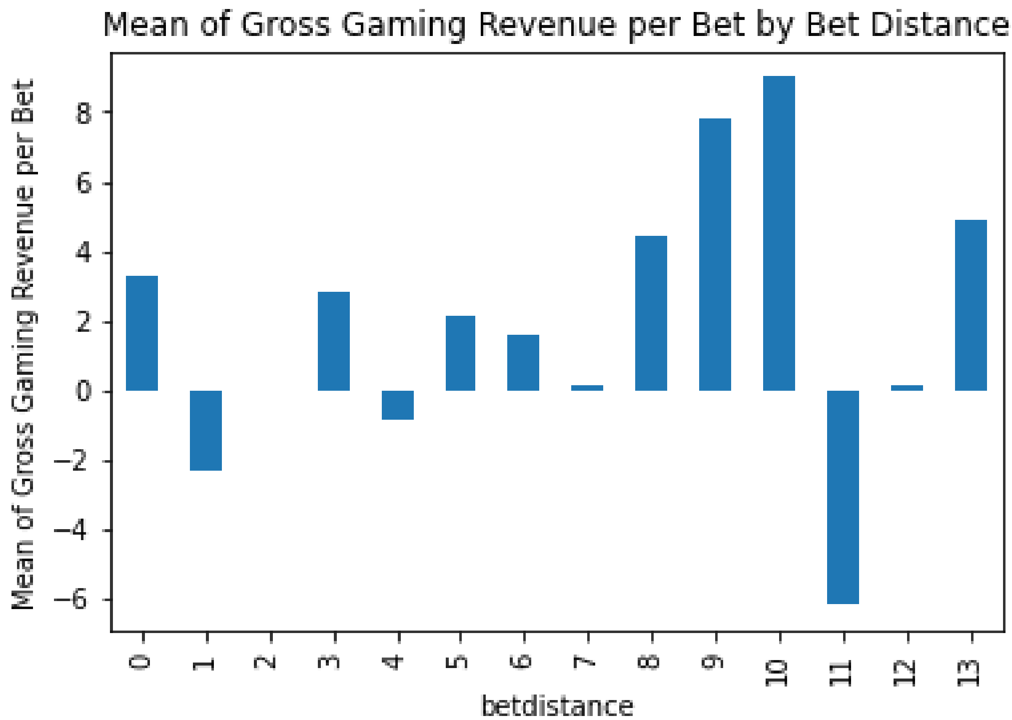
[26]:
```
mean_ggr_by_betdistance = cur_df.groupby('betdistance')['ggr'].mean()
mean_ggr_by_betdistance
```

[26]:
```
betdistance
0      3.300259
1     -2.307373
2     -0.007805
3      2.836584
4     -0.830632
5      2.096746
6      1.555140
7      0.097606
8      4.427661
9      7.822999
10     8.996325
11    -6.174307
12     0.144410
13     4.866736
Name: ggr, dtype: float64
```

[27]:
```
fig, ax = plt.subplots()

ax.set_title('Sum of Gross Gaming Revenue by Bet Distance')
ax.set_xlabel("Bet Distance")
ax.set_ylabel('Sum of Gross Gaming Revenue')
plt.xticks(rotation=45)
sum_ggr_by_betdistance.plot(kind='bar')

plt.show()
```




```
[28]: fig, ax = plt.subplots()

ax.set_title('Mean of Gross Gaming Revenue per Bet by Bet Distance')
ax.set_xlabel("Bet Distance")
ax.set_ylabel('Mean of Gross Gaming Revenue per Bet')

plt.xticks(rotation=45)
mean_ggr_by_betdistance.plot(kind='bar')

plt.show()
```




```
[29]: # its clear that theres a ton of revenue generated on same day bets. what about␣
      ↪combining all days before
```

```
[30]: # adjust for comp to be between day of vs 6 days before combined
```

```
[31]: '''3) Check how profitable future bets are for the book. Filter for bets placed␣
      ↪2+ weeks before settle date.
          What % of users are placing future bets.
          Actionable insight: If % is low and futures are profitable, push marketing␣
      ↪of future bets.
          Season-long futures are often marketed close to season open what if we have␣
      ↪week-long mid-season
          promotion on championship futures only.'''
```

```
[31]: '3) Check how profitable future bets are for the book. Filter for bets placed 2+
      weeks before settle date.\n    What % of users are placing future bets. \n
      Actionable insight: If % is low and futures are profitable, push marketing of
      future bets.\n    Season-long futures are often marketed close to season open
      what if we have week-long mid-season \n    promotion on championship futures
      only.'
```

```
[32]: # Get sum of GGR for cur_df and sum of GGR for f_df
      futures_ggr = f_df['ggr'].sum()
```

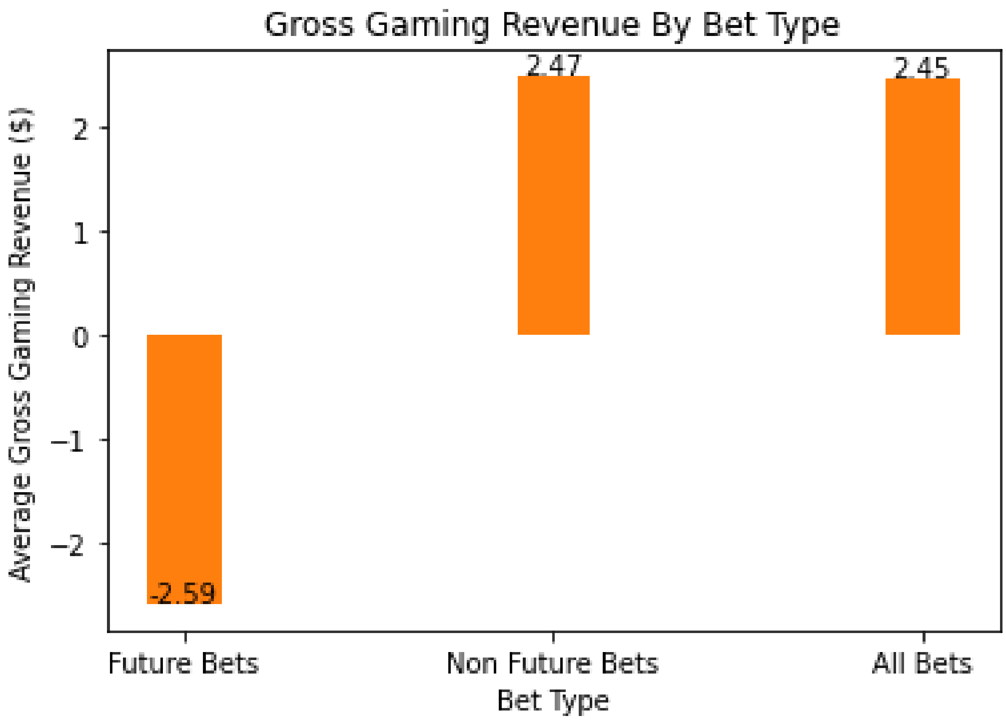
```python
reg_ggr = cur_df['ggr'].sum()
data = data_clean
total_ggr = data['ggr'].sum()
print(futures_ggr, reg_ggr, total_ggr)
```

```
-47588.778326999985 10281971.010207009 10234382.231879992
```

[33]:
```python
# now with average per bet
futures_ggr_mean = f_df['ggr'].mean()
reg_ggr_mean = cur_df['ggr'].mean()
data = data_clean
total_ggr_mean = data['ggr'].mean()
print(futures_ggr_mean, reg_ggr_mean, total_ggr_mean)
```

```
-2.5919813903594924 2.4739231619264817 2.4516426474746646
```

[34]:
```python
x = ["Future Bets", "Non Future Bets", "All Bets"]
y = [futures_ggr_mean, reg_ggr_mean, total_ggr_mean]

fig, ax = plt.subplots()

# creating the bar plot
plt.bar(x, y,
        width = 0.2)

bars = ax.bar(x, y, width=0.2)

# Function to annotate the bars
def annotate_bars(bars):
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, -1),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

# Annotate all bars
annotate_bars(bars)


ax.set_xlabel("Bet Type")
ax.set_ylabel("Average Gross Gaming Revenue ($)")
ax.set_title("Gross Gaming Revenue By Bet Type")
plt.show()
```



[35]:
```
'''4) Get playerids in top 10 of wagers place, total wagered, GGR to identify␣
↪power users.
    Actionable insight: Bonuses to top GGR players (biggest losers)'''
```

[35]: '4) Get playerids in top 10 of wagers place, total wagered, GGR to identify power users. \n   Actionable insight: Bonuses to top GGR players (biggest losers)'

[36]:
```
# Who are our volume power users?
data = data_clean

player_freq2 = data['playerid'].value_counts()
player_freq2 = player_freq2[player_freq2 > 10]
threshold_freq2 = player_freq2.quantile(0.10)
bot_players = player_freq2[player_freq2 <= threshold_freq2]

# Convert the player IDs to full numbers and then to strings
bot_players = bot_players.reset_index()
bot_players.columns = ['playerid', 'frequency']
bot_players['playerid'] = bot_players['playerid'].astype(int).astype(str)

print("99th percentile players by lowest frequency:")
print(bot_players)
```

```
99th percentile players by lowest frequency:
      playerid  frequency
0     25252770         15
1     23168951         15
2     17008635         15
3     34774908         15
4     25068732         15
…          …          …
2487  36029277         11
2488  34861159         11
2489  23847135         11
2490  35534551         11
2491  41028839         11

[2492 rows x 2 columns]
```

[37]:

```python
# Who are our volume weakest users?
data = data_clean

player_freq = data['playerid'].value_counts()
threshold_freq = player_freq.quantile(0.99)
top_players = player_freq[player_freq >= threshold_freq]

# Convert the player IDs to full numbers and then to strings
top_players = top_players.reset_index()
top_players.columns = ['playerid', 'frequency']
top_players['playerid'] = top_players['playerid'].astype(int).astype(str)

print("99th percentile players by frequency:")
print(top_players)
```

```
99th percentile players by frequency:
     playerid  frequency
0    22198825       3894
1     6744812       3794
2    22244887       3527
3    22647686       3321
4    35438248       3155
..        …          …
359  17493162       1215
360  23126494       1214
361  22688236       1214
362  23501970       1213
363   5464808       1212

[364 rows x 2 columns]
```

[38]:
```
# Group data by player ID and calculate average gross revenue
player_avg_ggr = data.groupby('playerid')['ggr'].mean()

player_avg_ggr.index = player_avg_ggr.index.astype(int).astype(str)
player_avg_ggr = player_avg_ggr.round(2)

data['avg_ggr'] = player_avg_ggr

threshold_avg_ggr = player_avg_ggr.quantile(0.99)
top_players_avg_ggr = player_avg_ggr[player_avg_ggr >= threshold_avg_ggr]
top_players_avg_ggr = top_players_avg_ggr.sort_values(ascending=False)

top_players_avg_ggr = top_players_avg_ggr.reset_index()

#data['in_top_players_avg_ggr'] = data['playerid'].astype(str).
 ↪isin(top_players_avg_ggr['playerid'])


print("Player IDs in the 99th percentile of average gross revenue, sorted by␣
 ↪average gross revenue:")
print(top_players_avg_ggr)
```

```
Player IDs in the 99th percentile of average gross revenue, sorted by average
gross revenue:
     playerid      ggr
0     5768578  4000.00
1    29140084  3666.60
2    41509016  2500.00
3    38987672  2296.52
4    30271610  2000.00
..        …        …
359  10149795   113.45
360  38160385   112.50
361  23543781   112.00
362  37240787   111.90
363  31175232   110.90

[364 rows x 2 columns]
```

[39]:
```
player_tot_ggr = data.groupby('playerid')['ggr'].sum()

player_tot_ggr.index = player_tot_ggr.index.astype(int).astype(str)
player_tot_ggr = player_tot_ggr.round(2)

threshold_tot_ggr = player_tot_ggr.quantile(0.99)
top_players_tot_ggr = player_tot_ggr[player_tot_ggr >= threshold_tot_ggr]
top_players_tot_ggr = top_players_tot_ggr.sort_values(ascending=False)
```

```
top_players_tot_ggr = top_players_tot_ggr.reset_index()

print("Player IDs in the 99th percentile of total gross revenue:")
print(top_players_tot_ggr)
```

```
Player IDs in the 99th percentile of total gross revenue:
     playerid       ggr
0    26100126  28891.45
1    23543907  20940.29
2     5457155  18234.24
3    25089725  17008.97
4    28650324  16744.38
..        …         …
359  17361506   5093.23
360  22238867   5092.24
361  32455040   5075.19
362  12202626   5069.41
363  38390449   5066.09

[364 rows x 2 columns]
```

[40]:

```
# 99th percentile average wager amount
player_avg_net_stake = data.groupby('playerid')['net_stake'].mean()

threshold_avg_net_stake = player_avg_net_stake.quantile(0.99)
top_players_wager_avg = player_avg_net_stake[player_avg_net_stake >=␣
 ↪threshold_avg_net_stake]

top_players_wager_avg.index = top_players_wager_avg.index.astype(int).
 ↪astype(str)

top_players_wager_avg = top_players_wager_avg.round(2)
top_players_wager_avg.columns = ['playerid', 'average_stake']
top_players_wager_avg = top_players_wager_avg.sort_values(ascending=False)
top_players_wager_avg = top_players_wager_avg.reset_index()

print("99th percentile of players by average wager:")
print(top_players_wager_avg)
```

```
99th percentile of players by average wager:
     playerid  net_stake
0    34801769    9414.33
1    38261481    5375.00
2    29140084    4292.33
3     5768578    4000.00
4    23740032    3593.38
..        …          …
```

```
359    702799     370.70
360  36044133     370.00
361  34180972     369.87
362  35761468     366.67
363  38200471     366.67

[364 rows x 2 columns]
```

[41]:
```python
# Sort players by total wagered and get the top 10
player_wager_sum = data.groupby('playerid')['net_stake'].sum()

threshold_wager_sum = player_wager_sum.quantile(0.99)
top_players_wager_sum = player_wager_sum[player_wager_sum >=␣
↪threshold_wager_sum]

top_players_wager_sum.index = top_players_wager_sum.index.astype(int).
↪astype(str)

top_players_wager_sum = top_players_wager_sum.round(2)
top_players_wager_sum.columns = ['playerid', 'total_wagers']
top_players_wager_sum = top_players_wager_sum.sort_values(ascending=False)
top_players_wager_sum = top_players_wager_sum.reset_index()


print("Top 10 players by total wagered:")
print(top_players_wager_sum)
```

```
Top 10 players by total wagered:
     playerid  net_stake
0     2102856  823527.79
1     2086282  472406.84
2    14257641  281766.91
3    35047561  264465.20
4    37182924  255582.88
..        …        …
359  24000107   36033.90
360  35199041   35989.00
361  25401112   35961.12
362  11848117   35768.77
363  25488700   35736.65

[364 rows x 2 columns]
```

[42]:
```python
bets_per_day_per_user = data.groupby(['playerid', 'placed_date']).size().
↪reset_index(name='bets_count')

total_bets_per_user = bets_per_day_per_user.groupby('playerid')['bets_count'].
↪sum()
```

```
total_days_per_user = bets_per_day_per_user.groupby('playerid')['placed_date'].
 ↪nunique()

average_bets_per_day_per_user = total_bets_per_user / total_days_per_user

threshold_abpd = average_bets_per_day_per_user.quantile(0.99)
top_abpd =␣
 ↪average_bets_per_day_per_user[average_bets_per_day_per_user>=threshold_abpd]
top_abpd.index = top_abpd.index.astype(int).astype(str)
top_abpd = top_abpd.sort_values(ascending=False)
top_abpd = top_abpd.reset_index()


print(top_abpd)
```

```
     playerid      0
0    11942318  171.0
1    37271793  106.0
2    33968238   83.0
3    37010747   70.3
4     9557872   54.0
..        …      …
367  11402873   17.0
368  19920227   17.0
369  26577373   17.0
370  25993602   17.0
371  19494432   17.0

[372 rows x 2 columns]
```

[43]:
```
# who bets on the most sport leagues
sports_per_user = data.groupby('playerid')['sportname'].nunique()
threshold_spu = sports_per_user.quantile(0.99)
top_players_spu = sports_per_user[sports_per_user>=threshold_spu]
top_players_spu.index = top_players_spu.index.astype(int).astype(str)
top_players_spu = top_players_spu.sort_values(ascending=False)
top_players_spu = top_players_spu.reset_index()
print(top_players_spu)
```

```
      playerid  sportname
0     37368793          7
1     19948082          7
2     20023294          7
3     20031314          7
4     20051691          7
…          …        …
1326  24651799          7
```

```
1327  24663637         7
1328  24666340         7
1329  24670007         7
1330     59716         7

[1331 rows x 2 columns]
```

[44]:
```python
# Calculate Average Retention Rate
time_window = '1M'  # 1 month

active_players = data.groupby(pd.Grouper(key='placed_date',␣
 ↪freq=time_window))['playerid'].unique()
initial_active_players = active_players.iloc[0]

#Calculate the retention for each time window
retention_rates = []
for i in range(1, len(active_players)):
    active_players_in_window = active_players.iloc[i]
    returning_players_count = len(set(initial_active_players) &␣
 ↪set(active_players_in_window))
    retention_rate = returning_players_count / len(initial_active_players)
    retention_rates.append(retention_rate)

    initial_active_players = active_players_in_window

average_retention_rate = sum(retention_rates) / len(retention_rates)

print("Average retention rate:", average_retention_rate)
```

```
Average retention rate: 0.6887276007133823
```

[45]:
```python
time_window = '1M'  # 1 month

active_players = data.groupby(['playerid', pd.Grouper(key='placed_date',␣
 ↪freq=time_window)])['wagerid'].count().unstack()

retention_rates = (active_players.notnull().sum(axis=1) / len(active_players.
 ↪columns))

data['retention'] = retention_rates

percentile_99 = retention_rates.quantile(0.99)

top_players_retention = retention_rates[retention_rates >= percentile_99]
top_players_retention.index = top_players_retention.index.astype(int).
 ↪astype(str)
top_players_retention = top_players_retention.to_frame()
top_players_retention = top_players_retention.reset_index()
```

```
print("Top players by retention rate:")
print(top_players_retention)
```

```
Top players by retention rate:
     playerid    0
0       59716  1.0
1       71364  1.0
2      118359  1.0
3      144943  1.0
4      314065  1.0
..        …    …
713  30967773  1.0
714  30968960  1.0
715  31000756  1.0
716  31018313  1.0
717  31286551  1.0

[718 rows x 2 columns]
```

[46]:
```
time_window = '1M'  # 1 month

active_players = data.groupby(['playerid', pd.Grouper(key='placed_date',␣
↪freq=time_window)])['wagerid'].count().unstack()

retention_rates = (active_players.notnull().sum(axis=1) / len(active_players.
↪columns))

percentile_20 = retention_rates.quantile(0.20)

p20_players_retention = retention_rates[retention_rates >= percentile_20]
p20_players_retention.index = p20_players_retention.index.astype(int).
↪astype(str)
p20_players_retention = p20_players_retention.to_frame()
p20_players_retention = p20_players_retention.reset_index()
```

[ ]:
```

```

[47]:
```
# Concatenate player IDs from all four dataframes
all_player_ids = pd.concat([top_players.playerid, top_players_avg_ggr.playerid,␣
↪top_players_wager_avg.playerid, top_players_wager_sum.playerid,␣
↪top_players_tot_ggr.playerid,
                           top_players_retention.playerid,
                            top_players_spu.playerid,
                            top_abpd.playerid,])

# Count occurrences of each player ID
```

```
player_id_counts = all_player_ids.value_counts()

# Sort player IDs based on their appearance count
ranked_player_ids = player_id_counts.sort_values(ascending=False)

# Display the result
print("Player IDs ranked by how many 99th percentile dataframes they appear in:
↪")
#print(ranked_player_ids.head(50))
```

```
Player IDs ranked by how many 99th percentile dataframes they appear in:
```

[48]:
```
eliteids = pd.concat([top_players_tot_ggr.playerid,
                           top_players_retention.playerid,
                            top_players_spu.playerid,
                            top_abpd.playerid,])
id_counts = eliteids.value_counts()
ranked_elite_ids = id_counts.sort_values(ascending=False)
#print("ELITE USER IDS:")
#print(ranked_elite_ids.head(30))
#ranked_elite_ids.head(30).to_csv("eliteusers.csv")
id_counts
```

[48]:
```
22200251    3
30599220    3
11265689    3
24933696    3
22260231    3
           ..
27489019    1
34207883    1
30461534    1
20545738    1
41185627    1
Name: playerid, Length: 2418, dtype: int64
```

[49]:
```
#from tabulate import tabulate
power_users = ranked_player_ids.head(46)
#print(tabulate(power_users, headers='keys', tablefmt='pretty'))

print(power_users.to_string())
```

```
17097245    5
15577824    5
22299023    5
22200251    5
12646659    5
22244887    5
```

```
6744812     5
25129084    4
22290305    4
19571704    4
11265689    4
12910654    4
3213832     4
1273437     4
18763877    4
1564786     4
23762354    4
22310589    4
38788054    4
24248275    4
20639546    4
6121865     4
22260231    4
28650324    4
20407988    4
4598502     4
22197979    4
22688236    4
39791755    4
27322267    4
34625329    4
25378683    4
22198825    4
26100126    4
6592117     4
23078824    4
22201558    4
24933696    4
24302472    4
27837505    4
13366089    4
8047354     4
27066598    4
2102856     4
22864328    4
22648433    4
```

[50]:

```
# targeted users 1
# Users with high average revenue (GGR) per bet placed and active* low wager␣
↪frequency.
# *Filtered for users with 10+ bets who are also in the 20th percentile of␣
↪retention rate
# to avoid using inactive users
```

[51]:
```
time_window = '1M'
active_players = data.groupby(
    ['playerid', pd.Grouper(key='placed_date', freq=time_window)])['wagerid'].
 ↪count().unstack()

retention_rates = (active_players.notnull().sum(axis=1) / len(active_players.
 ↪columns))

bet_counts = data['playerid'].value_counts()
filtered_players = bet_counts[bet_counts >= 10]

percentile_5 = retention_rates.quantile(0.05)

percentile_10_bet_counts = filtered_players.quantile(0.10)

avg_ggr_per_player = data.groupby('playerid')['ggr'].mean()

percentile_95_avg_ggr = avg_ggr_per_player.quantile(0.95)

# Get player IDs that meet all criteria
selected_players = retention_rates[(retention_rates >= percentile_5) &␣
 ↪(bet_counts <= percentile_10_bet_counts)]
selected_players = selected_players[
    selected_players.index.isin(avg_ggr_per_player[avg_ggr_per_player >=␣
 ↪percentile_95_avg_ggr].index)]
selected_players.index = selected_players.index.astype(int).astype(str)

targets = selected_players.index.tolist()
```

[52]:
```
print(len(targets))
print("Player IDs meeting the criteria:")
#print(selected_players.index.tolist())
```

```
1364
Player IDs meeting the criteria:
```

[53]:
```
'''5) Check correlation between sportname and GGR. Can filter for only straight␣
 ↪bets as parlays likely strongly correlate with high GGR.
'''
```

[53]:
```
'5) Check correlation between sportname and GGR. Can filter for only straight
bets as parlays likely strongly correlate with high GGR.\n'
```

[54]:
```
# Check correlation between sport & ggr for all bets, straight bets & parlays
data = data_clean

# Perform label encoding on 'sportname'
```

```
label_encoder = LabelEncoder()
data['sportname_encoded'] = label_encoder.fit_transform(data['sportname'])

correlation = data['sportname_encoded'].corr(data['ggr'])

print("Correlation between 'sportname' and 'ggr' for all bets:", correlation)
```

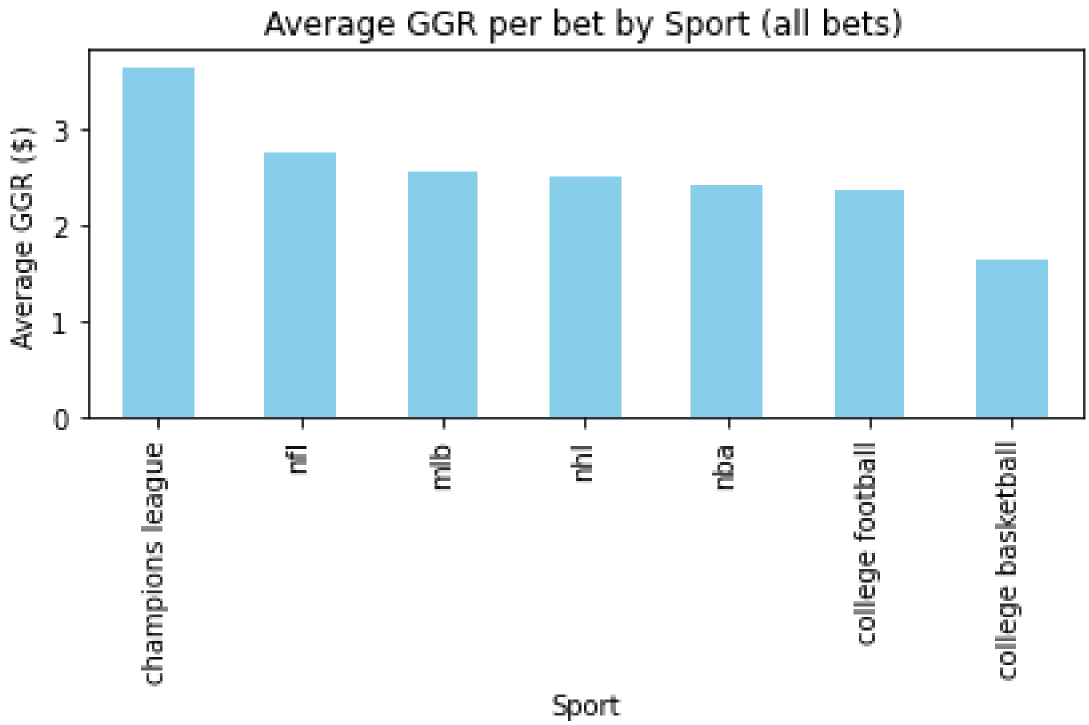
```
Correlation between 'sportname' and 'ggr' for all bets: 0.0030504786163370896
```

[55]:
```
data_list = [data_clean, straight_bets, parlay_bets]
```

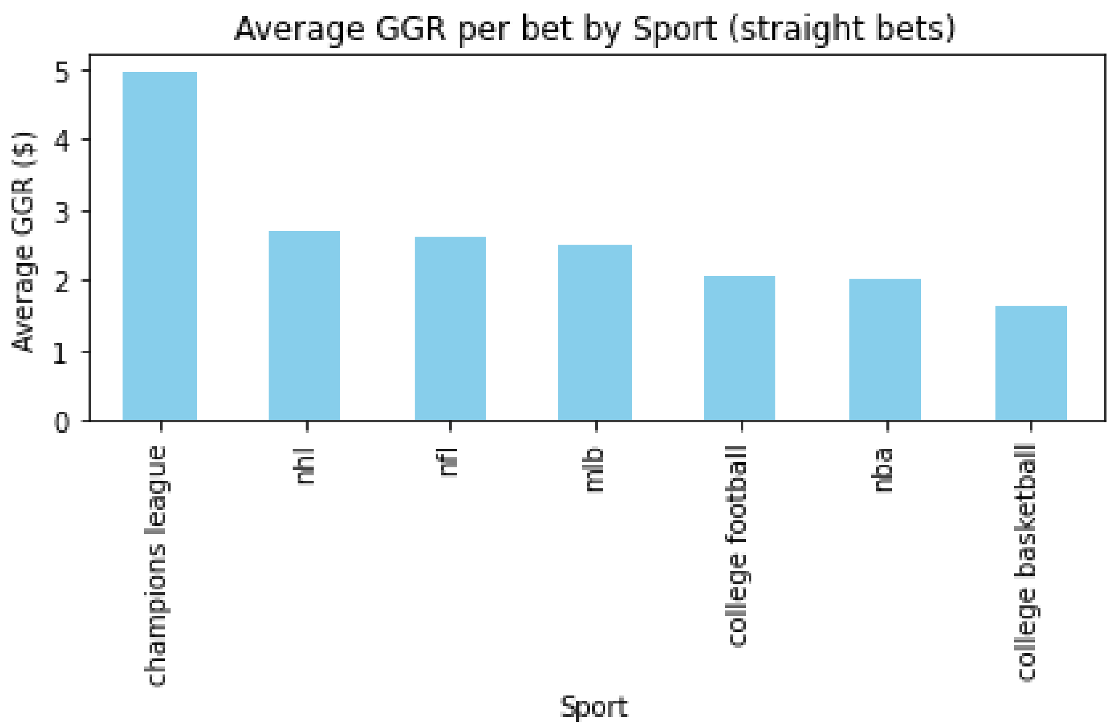
[56]:
```
i = 0
for data in data_list:
    #data['ggr'] = data['ggr'].apply(lambda x: round(float(x), 2))

    avg_ggr_per_sport = data.groupby('sportname')['ggr'].mean().
↪sort_values(ascending=False)
    print(avg_ggr_per_sport)

    fig, ax = plt.subplots()
    avg_ggr_per_sport.plot(kind='bar', color='skyblue')

    if i == 0:
        option = "(all bets)"
    elif i == 1:
        option = "(straight bets)"
    elif i == 2:
        option = "(parlay bets)"
    else:
        raise Exception("Invalid bet type")

    i+=1

    ax.set_title("Average GGR per bet by Sport "+option)
    ax.set_xlabel("Sport")


    ax.set_ylabel("Average GGR ($)")
    plt.tight_layout()
    plt.savefig("Average_GGR_per_bet_by_Sport_{option}.png".
↪format(option=option),transparent=True)

    plt.show()
```

```
sportname
champions league      3.639906
nfl                   2.742356
mlb                   2.559210
```

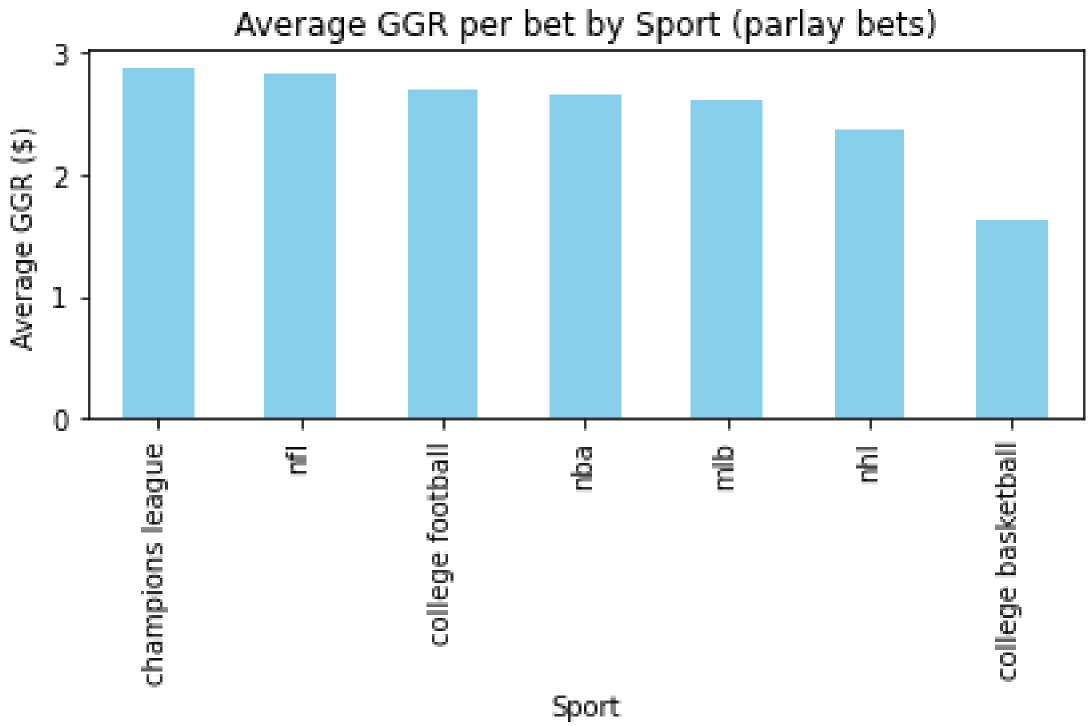
```
nhl                     2.510377
nba                     2.407713
college football        2.369812
college basketball      1.637004
Name: ggr, dtype: float64
```




```
sportname
champions league        4.965027
nhl                     2.703385
nfl                     2.613300
mlb                     2.505624
college football        2.053449
nba                     2.013292
college basketball      1.643920
Name: ggr, dtype: float64
```

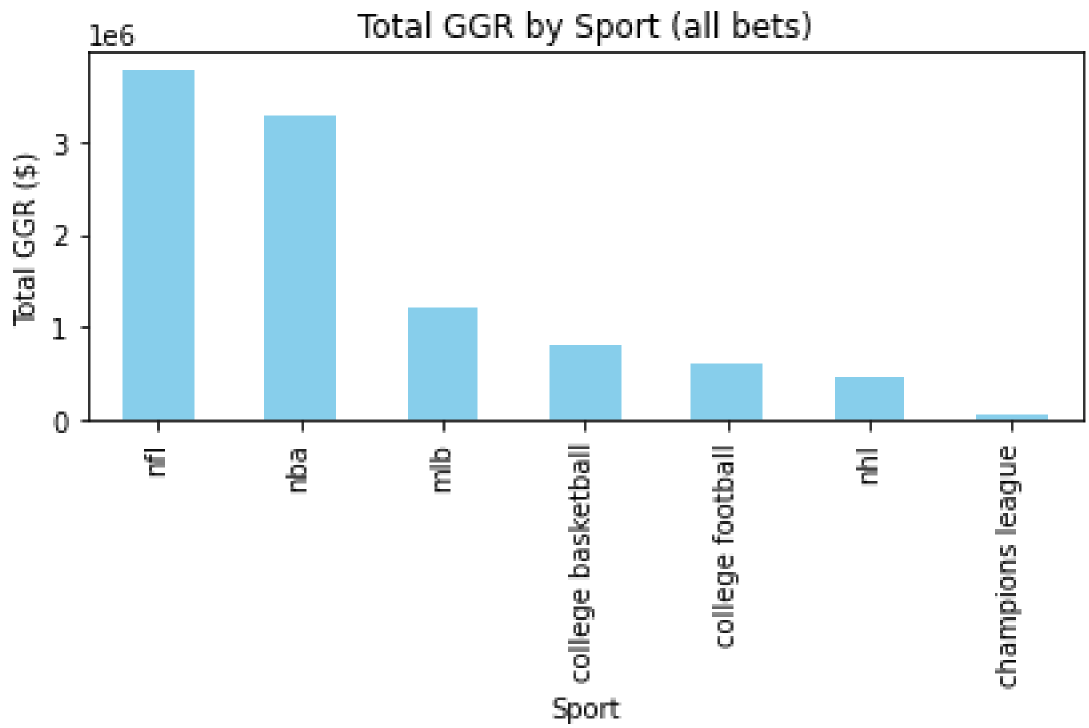


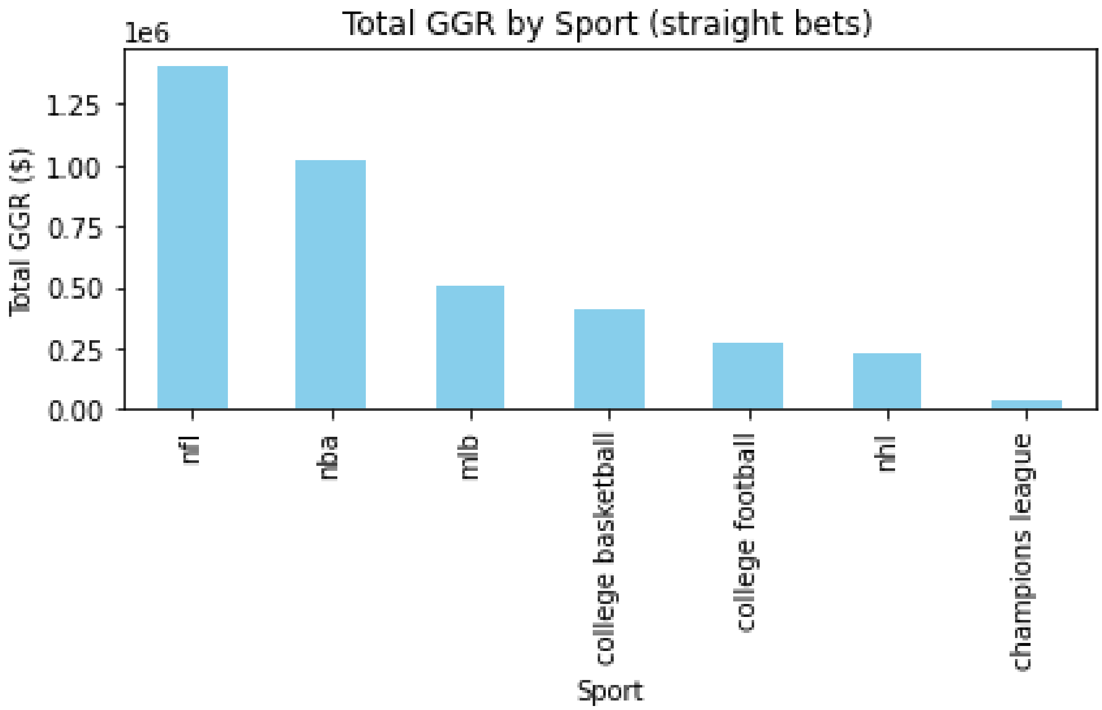

```
sportname
champions league      2.869291
nfl                   2.825278
college football      2.685386
nba                   2.641652
mlb                   2.598111
nhl                   2.358641
college basketball    1.629941
Name: ggr, dtype: float64
```

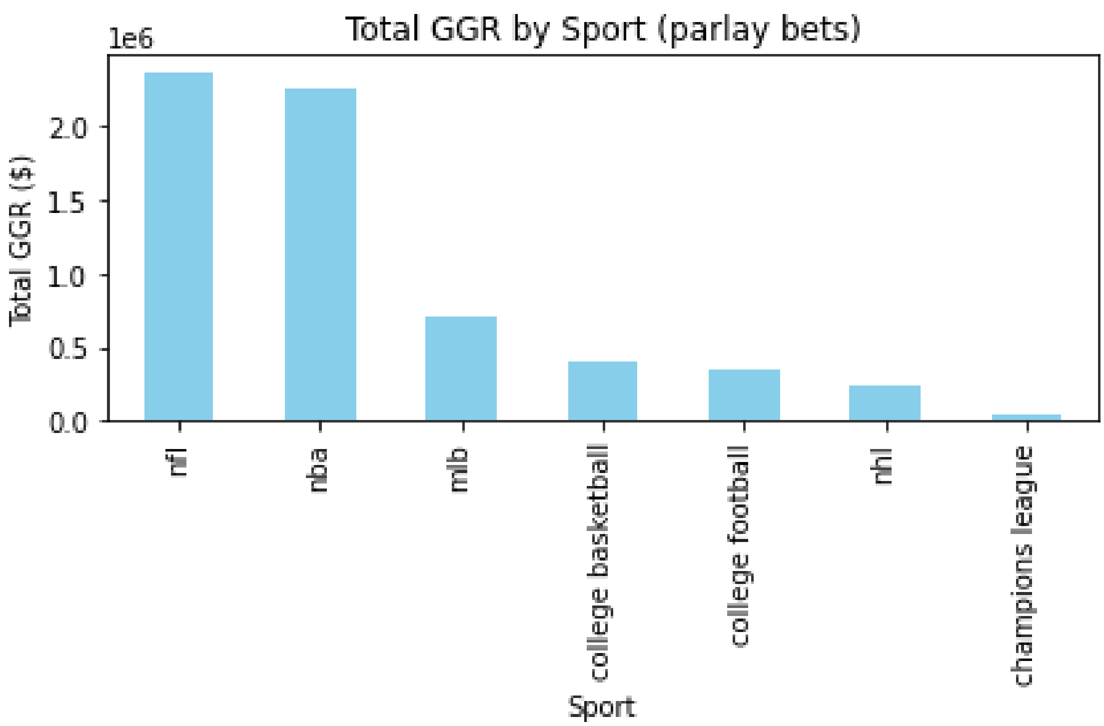


[57]:
```
i = 0
for data in data_list:
    #data['ggr'] = data['ggr'].apply(lambda x: round(x, 2))
    total_ggr_per_sport = data.groupby('sportname')['ggr'].sum().
 ↪sort_values(ascending=False)
    #print(total_ggr_per_sport)

    fig, ax = plt.subplots()
    total_ggr_per_sport.plot(kind='bar', color='skyblue')

    if i == 0:
        option = "(all bets)"
    elif i == 1:
        option = "(straight bets)"
    elif i == 2:
        option = "(parlay bets)"
    else:
        raise Exception("Invalid bet type")

    i+=1

    ax.set_title("Total GGR by Sport "+option)
    ax.set_xlabel("Sport")
```

```
    ax.set_ylabel("Total GGR ($)")
    plt.tight_layout()
    plt.savefig("Total_GGR_per_bet_by_Sport_{option}.png".
↪format(option=option),transparent=True)


    plt.show()
```




Total GGR by Sport (straight bets)
1e6
Total GGR ($)
1.25
1.00
0.75
0.50
0.25
0.00
nfl
nba
mlb
college basketball
college football
nhl
champions league
Sport


Total GGR by Sport (parlay bets)
1e6
Total GGR ($)
2.0
1.5
1.0
0.5
0.0
nfl
nba
mlb
college basketball
college football
nhl
champions league
Sport

[58]:
```
'''compare sports regular season to playoff growth
all sports should grow similarly, at very least grow'''
```

[58]: 'compare sports regular season to playoff growth\nall sports should grow similarly, at very least grow'

[59]:
```
'''
data range: 2021-03-28 --> 2022-03-29

nfl: 2021-09-09 --> 2022-01-09
nfl playoff: 2022-01-15 --> 2022-02-13

nba: 2021-10-19 --> end of dataset 2022-03-29
nba playoff (year before): 2021-05-22 --> 2021-06-20

mlb: 2021-04-21 --> 2021-10-04
mlb playoff: 2021-10-05 --> 2021-11-02

ncaab: 2021-11-09 --> 2021-03-13
ncaab tournament: 2021-03-18 --> 2021-04-05

ncaaf: 2021-08-28 --> 2021-12-11
ncaaf bowl season: 2021-12-17 --> 2022-01-10

nhl: dataset start 2021-03-28 --> 2021-05-14
nhl playoff: 2021-05-15 --> 2021-07-07

cl qual: 2021-06-22 --> 2021-08-25
cl comp: 2021-09-14 --> end of dataset 2022-03-29
cl comp (year prior): start of dataset 2021-03-28 --> 2021-05-29
'''
```

[59]: '\ndata range: 2021-03-28 --> 2022-03-29\n\nnfl: 2021-09-09 --> 2022-01-09\nnfl playoff: 2022-01-15 --> 2022-02-13\n\nnba: 2021-10-19 --> end of dataset 2022-03-29\nnba playoff (year before): 2021-05-22 --> 2021-06-20\n\nmlb: 2021-04-21 --> 2021-10-04\nmlb playoff: 2021-10-05 --> 2021-11-02\n\nncaab: 2021-11-09 --> 2021-03-13\nncaab tournament: 2021-03-18 --> 2021-04-05\n\nncaaf: 2021-08-28 --> 2021-12-11\nncaaf bowl season: 2021-12-17 --> 2022-01-10\n\nnhl: dataset start 2021-03-28 --> 2021-05-14\nnhl playoff: 2021-05-15 --> 2021-07-07\n\ncl qual: 2021-06-22 --> 2021-08-25\ncl comp: 2021-09-14 --> end of dataset 2022-03-29\ncl comp (year prior): start of dataset 2021-03-28 --> 2021-05-29\n'

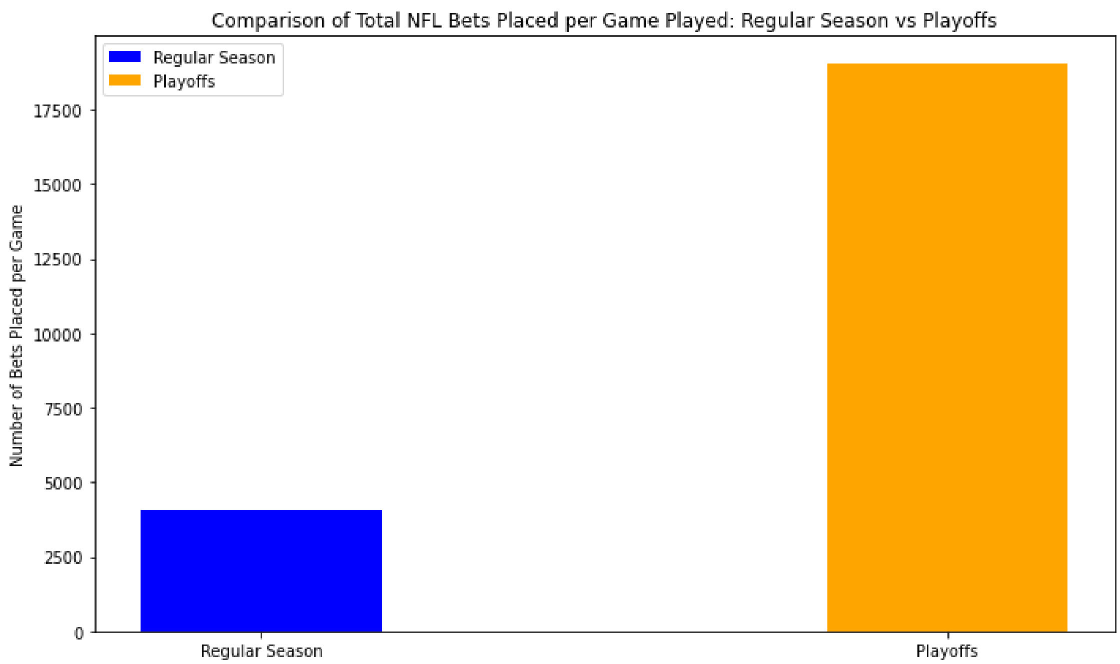
[103]:
```
# remove future bets
data = cur_df
start_date_range1 = '2021-09-09'
end_date_range1 = '2022-01-09'
```

```
start_date_range2 = '2022-01-15'
end_date_range2 = '2022-02-13'

nfl_reg = data[(data['sportname'] == 'nfl') &
                              (data['settled_date'] >= start_date_range1) &
                              (data['settled_date'] <= end_date_range1)]

nfl_playoff = data[(data['sportname'] == 'nfl') &
                              (data['settled_date'] >= start_date_range2) &
                              (data['settled_date'] <= end_date_range2)]

total_bets_reg = len(nfl_reg.index)
total_bets_pl = len(nfl_playoff.index)
print(total_bets_reg)
print(total_bets_pl)
```

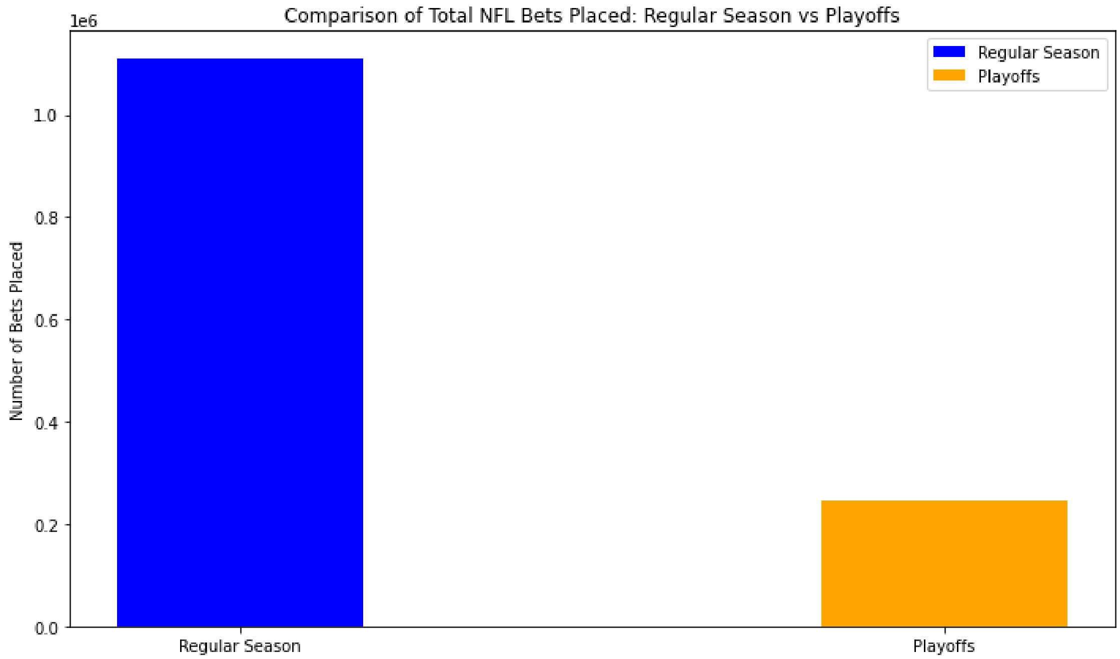
```
1108401
247263
```

[95]:

```
# bets per game
fig, ax = plt.subplots(figsize=(10, 6))

bar_width = 0.35
index = [1, 2]

bars1 = ax.bar(index[0], total_bets_reg/272, bar_width, label='Regular Season',␣
 ↪color='blue')
bars2 = ax.bar(index[1], total_bets_pl/13, bar_width, label='Playoffs',␣
 ↪color='orange')

#ax.set_xlabel('Date Range')
ax.set_ylabel('Number of Bets Placed per Game')
ax.set_title('Comparison of Total NFL Bets Placed per Game Played: Regular␣
 ↪Season vs Playoffs')
ax.set_xticks(index)
ax.set_xticklabels(['Regular Season', 'Playoffs'])
ax.legend()

plt.tight_layout()
#plt.savefig("Total NFL Bets Placed per Game Played: Regular Season vs Playoffs.
 ↪png",transparent=True)
plt.show()
```



[97]:
```
# total
fig, ax = plt.subplots(figsize=(10, 6))

bar_width = 0.35
index = [1, 2]

bars1 = ax.bar(index[0], total_bets_reg, bar_width, label='Regular Season',␣
 ↪color='blue')
bars2 = ax.bar(index[1], total_bets_pl, bar_width, label='Playoffs',␣
 ↪color='orange')

#ax.set_xlabel('Date Range')
ax.set_ylabel('Number of Bets Placed')
ax.set_title('Comparison of Total NFL Bets Placed: Regular Season vs Playoffs')
ax.set_xticks(index)
ax.set_xticklabels(['Regular Season', 'Playoffs'])
ax.legend()

plt.tight_layout()
#plt.savefig("Total NFL Bets Placed: Regular Season vs Playoffs.
 ↪png",transparent=True)
plt.show()
```



[ ]:

```

```

[104]:

```
playoff_growth_pg = (total_bets_pl/13)/(total_bets_reg/272)-1
print("NFL regular season to playoff growth (bets per game) is␣
↪",round((playoff_growth_pg)*100,1),"%")
```

```
NFL regular season to playoff growth (bets per game) is  366.8 %
```

[105]:

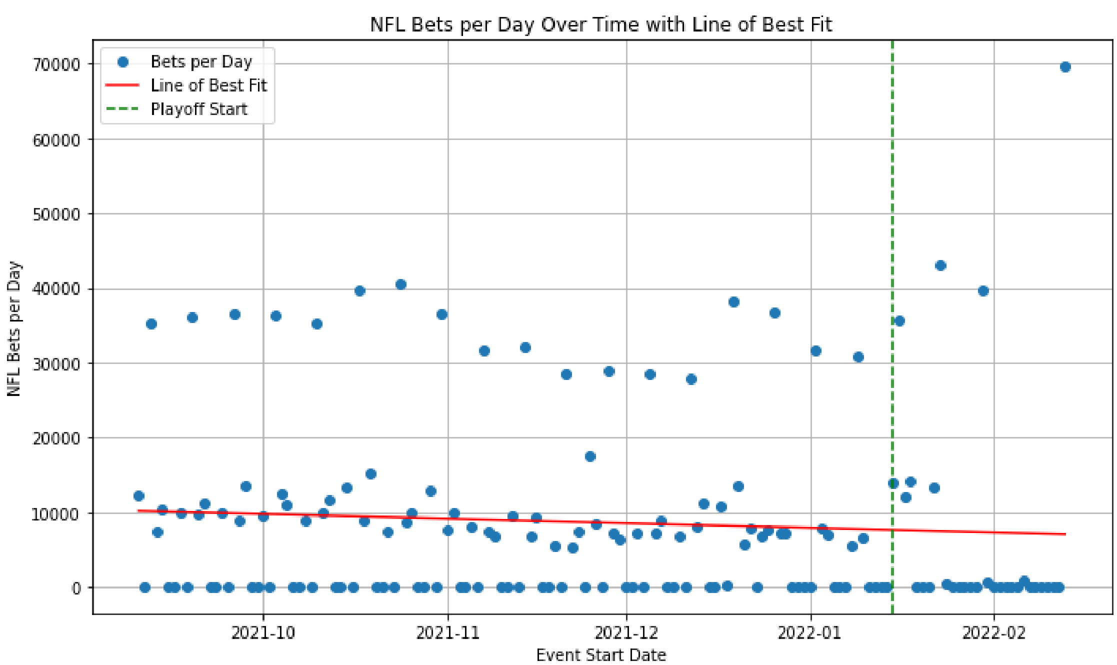
```
playoff_growth = (total_bets_pl)/(total_bets_reg)-1
print("NFL regular season to playoff growth (bets total) is␣
↪",round((playoff_growth)*100,1),"%")
```

```
NFL regular season to playoff growth (bets total) is  -77.7 %
```

[62]:

```
start_date = '2021-09-09'
end_date = '2022-02-13'

nfl_bets_range = data[(data['sportname'] == 'nfl') &
                            (data['event_start_date'] >= start_date) &
                            (data['event_start_date'] <= end_date)]

# Step 2: Group the filtered data by settled_date and calculate the average␣
↪number of bets per day
#nfl_bets_range = nfl_bets_range[nfl_bets_range['net_stake'] >= 5000]
```

```
bets_per_day = nfl_bets_range.groupby('event_start_date').size().resample('D').
↪sum().reset_index()

# Step 3: Fit a linear regression line to the data
X = bets_per_day.index.values.reshape(-1, 1)  # Reshape to a column vector
y = bets_per_day.values[:, 1]  # Get the average bets per day as y values
regressor = LinearRegression()
regressor.fit(X, y)

# Step 4: Plot the average user bets per day against settled_date and the line␣
↪of best fit
fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(bets_per_day['event_start_date'], bets_per_day.values[:, 1],␣
↪marker='o', linestyle='', label='Bets per Day')
ax.plot(bets_per_day['event_start_date'], regressor.predict(X), color='red',␣
↪linestyle='-', label='Line of Best Fit')
ax.axvline(x='2022-01-15', color='green', linestyle='--', label='Playoff Start')

ax.set_xlabel('Event Start Date')
ax.set_ylabel('NFL Bets per Day')
ax.set_title('NFL Bets per Day Over Time with Line of Best Fit')
ax.grid(True)
ax.legend()

plt.tight_layout()
plt.show()
```



[63]:
```
'''NHL
nhl: dataset start 2021-03-28 --> 2021-05-14
nhl playoff: 2021-05-15 --> 2021-07-07'''
```

[63]: 'NHL\nnhl: dataset start 2021-03-28 --> 2021-05-14\nnhl playoff: 2021-05-15 -->
2021-07-07'

[64]:
```
# remove future bets
data = cur_df

start_date_range1 = '2021-03-28'
end_date_range1 = '2021-05-14'
start_date_range2 = '2021-05-15'
end_date_range2 = '2021-07-07'

nhl_reg = data[(data['sportname'] == 'nhl') &
                             (data['settled_date'] >= start_date_range1) &
                             (data['settled_date'] <= end_date_range1)]

nhl_playoff = data[(data['sportname'] == 'nhl') &
                             (data['settled_date'] >= start_date_range2) &
                             (data['settled_date'] <= end_date_range2)]

total_bets_nhl_reg = len(nhl_reg.index)
total_bets_nhl_pl = len(nhl_playoff.index)

# there were 340 regular season nhl games played in our range per␣
 ↪hockey-reference.com

#BPG = bets per game
bpg_nhl_reg = total_bets_nhl_reg/340
bpg_nhl_pl = total_bets_nhl_pl/84 # there were 84 total games in the 2021 NHL␣
 ↪Playoffs

print(bpg_nhl_reg, bpg_nhl_pl)
```

51.61764705882353 178.79761904761904

[65]:
```
'''
NCAAF
ncaaf: 2021-08-28 --> 2021-12-11
ncaaf bowl season: 2021-12-17 --> 2022-01-10
'''
```

[65]: '\nNCAAF\nncaaf: 2021-08-28 --> 2021-12-11\nncaaf bowl season: 2021-12-17 -->
2022-01-10\n'

[66]:
```
# remove future bets
data = cur_df

start_date_range1 = '2021-08-28'
end_date_range1 = '2021-12-11'
start_date_range2 = '2021-12-17'
end_date_range2 = '2022-01-10'

ncaaf_reg = data[(data['sportname'] == 'college football') &
                             (data['settled_date'] >= start_date_range1) &
                             (data['settled_date'] <= end_date_range1)]

ncaaf_bowl = data[(data['sportname'] == 'college football') &
                             (data['settled_date'] >= start_date_range2) &
                             (data['settled_date'] <= end_date_range2)]

total_bets_ncaaf_reg = len(ncaaf_reg.index)
total_bets_ncaaf_bowl = len(ncaaf_bowl.index)


# How many regular season games?
# 133 FBS teams * 12 games / 2 teams per game = 798
# I dont believe this number reflects the data since it doesnt take into␣
↪account FCS vs FBS matchups
# Will have to not use BPG for NCAAF

#BPG = bets per game
bpg_ncaaf_reg = total_bets_ncaaf_reg/1
bpg_ncaaf_bowl = total_bets_ncaaf_bowl/38

print(bpg_ncaaf_reg, bpg_ncaaf_bowl)
```

239757.0 540.578947368421

[67]:
```
champions_league_users = data[data['sportname'] == 'Champions␣
↪League']['playerid'].unique()

wagers_per_day = data.groupby(['playerid', pd.Grouper(key='placed_date',␣
↪freq='D')])['net_stake'].sum().reset_index()

top_wagers_per_day_users = wagers_per_day.groupby('playerid')['net_stake'].
↪sum().nlargest(1000).index
```

```
data.loc[:, 'champions_league_bet'] = data['playerid'].
↪isin(champions_league_users).astype(int)
data.loc[:, 'top_wagers_per_day'] = data['playerid'].
↪isin(top_wagers_per_day_users).astype(int)

correlation = data['champions_league_bet'].corr(data['top_wagers_per_day'])

print("Correlation coefficient:", correlation)
```

```
/Users/jeremysmith/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1596: 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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = _infer_fill_value(value)
/Users/jeremysmith/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1745: 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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(ilocs[0], value)
```

```
Correlation coefficient: nan
```

[86]:
```
champions_league_users = data[data['sportname'] == 'champions␣
↪league']['playerid'].unique()

bets_per_player = data.groupby('playerid')['wagerid'].count()

has_champions_league_bet = bets_per_player.index.isin(champions_league_users)

avg_bets_with_champions_league = bets_per_player[has_champions_league_bet].
↪mean()
avg_bets_without_champions_league = bets_per_player[~has_champions_league_bet].
↪mean()


# Check if players with at least one Champions League bet rank higher in volume␣
↪of bets placed
if avg_bets_with_champions_league > avg_bets_without_champions_league:
    print("Players with at least one Champions League bet rank higher in volume␣
↪of bets placed.")
elif avg_bets_with_champions_league < avg_bets_without_champions_league:
```

```
    print("Players with zero Champions League bets rank higher in volume of␣
 ↪bets placed.")
else:
    print("Players with and without Champions League bets have the same average␣
 ↪of bets placed.")
print(avg_bets_with_champions_league)
print(avg_bets_without_champions_league)
```

```
Players with at least one Champions League bet rank higher in volume of bets
placed.
340.567925137141
93.69983177120885
```

[ ]:
```
user_sport_counts = data.groupby('playerid')['sportname'].nunique()

users_with_2_or_fewer_sports = user_sport_counts[user_sport_counts <= 2]

num_users = len(users_with_2_or_fewer_sports)

print("Number of users who have never bet on more than 2 different sport types:
 ↪", num_users)
```

[ ]:
```
```