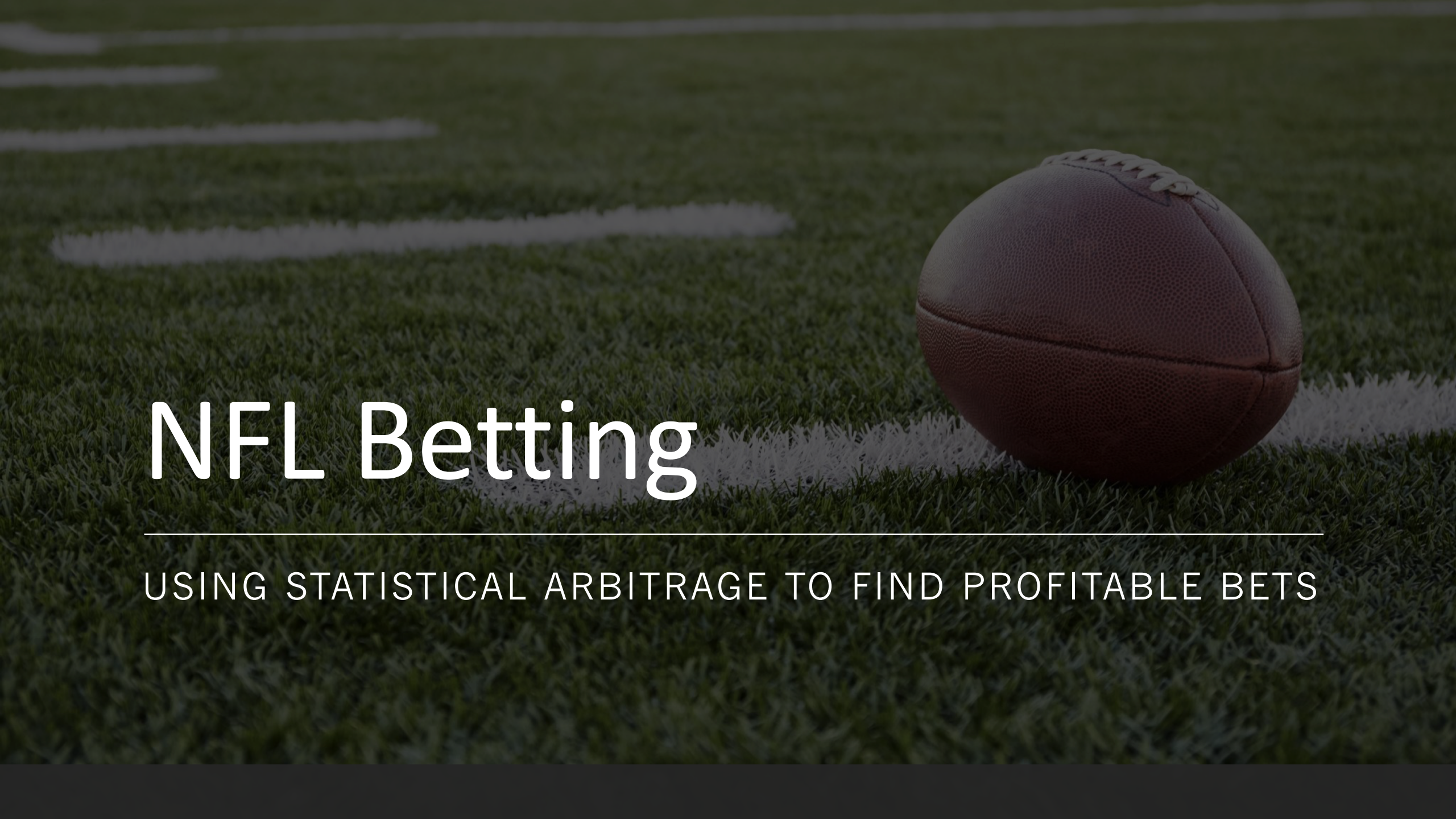


NFL Betting

USING STATISTICAL ARBITRAGE TO FIND PROFITABLE BETS

$$EV = (\alpha)(a_1 f_1 + a_2 f_2) + \beta a_1 f_1 + \gamma a_2 f_2 - f_1 - f_2$$

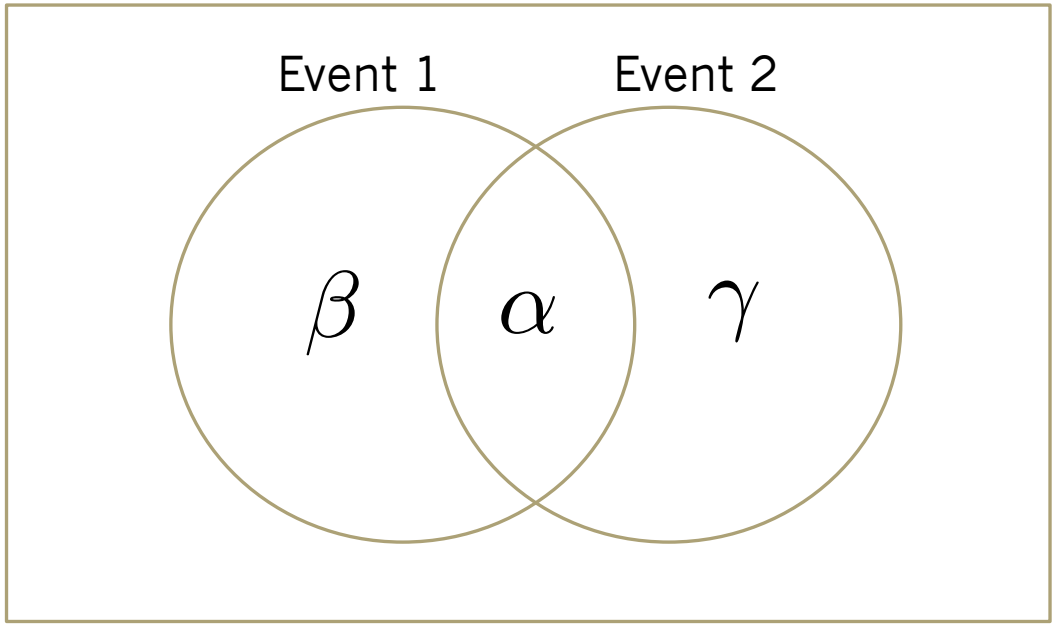
$\alpha = Probability\ of\ events\ 1\ and\ 2\ occurring$
$\beta = Probability\ of\ event\ 1\ and\ not\ event\ 2$
$\gamma = Probability\ of\ event\ 2\ and\ not\ event\ 1$

$f_1 = Amount\ bet\ on\ event\ 1$
$f_2 = Amount\ bet\ on\ event\ 2$

$m_1 = American\ odds\ for\ event\ 1$
$m_2 = American\ odds\ for\ event\ 2$

$$a_1 = \begin{cases} \frac{100+m_1}{m_1} & if\ favored \\ \frac{100+m_1}{100} & if\ underdogs \end{cases}$$

$$a_2 = \begin{cases} \frac{100+m_2}{m_2} & if\ favored \\ \frac{100+m_2}{100} & if\ underdogs \end{cases}$$



FinalNotebook

October 15, 2019

1 NFL Betting: Using Statistical Arbitrage To Make Profitable Bets

In [70]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 15 01:28:51 2019

@author: rajdua
"""



import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.feature_selection import RFE
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import brier_score_loss, roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.calibration import CalibratedClassifierCV as CCV
from sklearn import model_selection
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, VotingClassi
from sklearn.calibration import CalibratedClassifierCV as CCV
from sklearn import preprocessing
```

1.1 Initial Data Exploration and Pre-Processing

In [71]:
```
betting = pd.read_csv("nfl-scores-and-betting-data/spreadspoke_scores.csv")
qb = pd.read_csv("nflstatistics/Game_Logs_Quarterback.csv")
```

```
betting['schedule_date'] = pd.to_datetime(betting['schedule_date'])
```

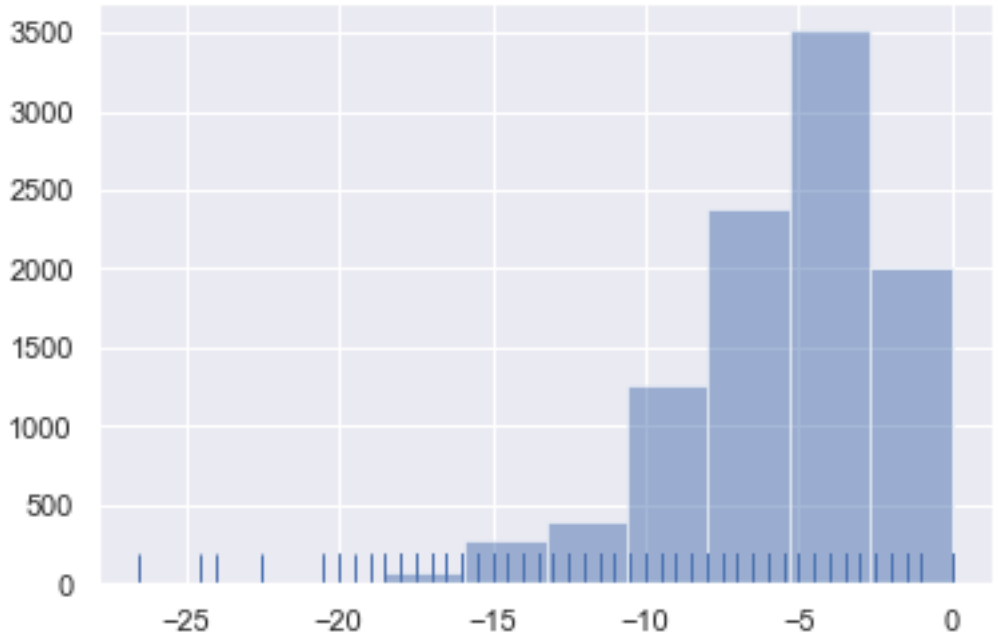
```
In [72]: print(betting.head())
         betting.isnull().sum()
         betting = betting.dropna(subset  = ['spread_favorite', 'over_under_line'])
```

```
  schedule_date  schedule_season schedule_week  schedule_playoff  \
0    1966-09-02             1966             1             False
1    1966-09-03             1966             1             False
2    1966-09-04             1966             1             False
3    1966-09-09             1966             2             False
4    1966-09-10             1966             1             False

            team_home  score_home  score_away        team_away  \
0      Miami Dolphins        14.0        23.0  Oakland Raiders
1      Houston Oilers        45.0         7.0   Denver Broncos
2  San Diego Chargers        27.0         7.0    Buffalo Bills
3      Miami Dolphins        14.0        19.0    New York Jets
4   Green Bay Packers        24.0         3.0  Baltimore Colts

  team_favorite_id  spread_favorite over_under_line         stadium  \
0              NaN              NaN             NaN     Orange Bowl
1              NaN              NaN             NaN    Rice Stadium
2              NaN              NaN             NaN  Balboa Stadium
3              NaN              NaN             NaN     Orange Bowl
4              NaN              NaN             NaN   Lambeau Field

   stadium_neutral  weather_temperature  weather_wind_mph weather_humidity  \
0            False                 83.0               6.0               71
1            False                 81.0               7.0               70
2            False                 70.0               7.0               82
3            False                 82.0              11.0               78
4            False                 64.0               8.0               62

  weather_detail
0            NaN
1            NaN
2            NaN
3            NaN
4            NaN
```

1.1.1 Spread distribution

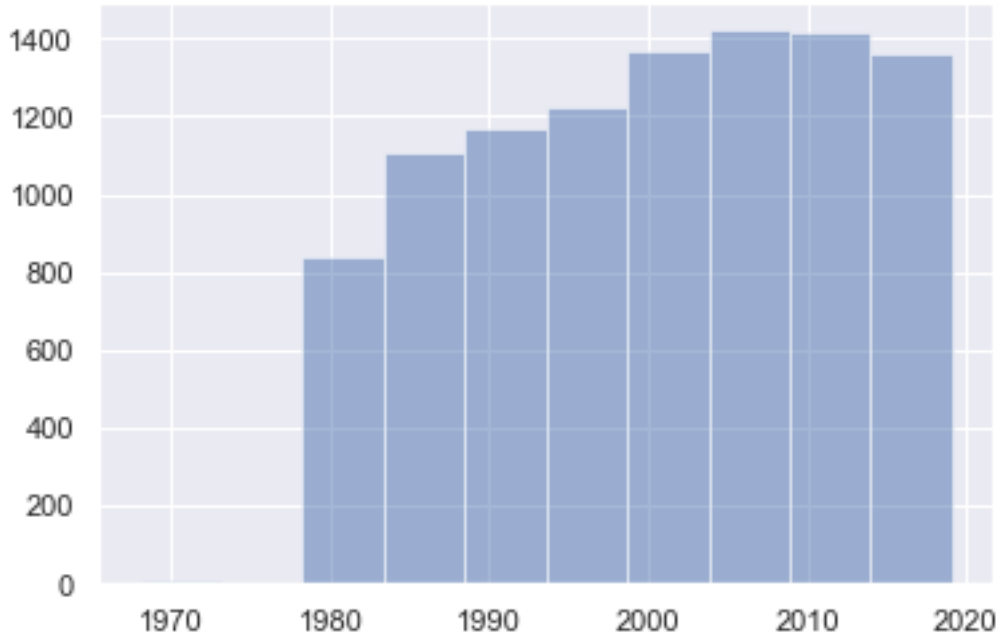
```
In [73]: plt.figure(0)
         plt.hist(betting.spread_favorite, alpha=0.5)
         sns.rugplot(betting.spread_favorite)
```

```
Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1fa92e80>
```



1.1.2 Year Distribution

In [74]:
```
plt.figure(1)
plt.hist(betting.schedule_date, alpha= 0.5)

teams = pd.read_csv("nfl-scores-and-betting-data/nfl_teams.csv")
teams = teams[["team_name", "team_id"]]
team_dict = dict(zip(teams.team_name, teams.team_id))
betting = betting.replace({"team_home" : team_dict})
betting = betting.replace({"team_away" : team_dict})
```




```
In [75]: def favored(x):
             if (x['team_home'] == x['team_favorite_id']):
                 return True
             else:
                 return False

         betting['spread_favorite'] = betting['spread_favorite'] * -1

         betting['home_favorite'] = betting.apply(lambda x: favored(x), axis = 1)
         betting = betting[betting["over_under_line"] != ' ']
         betting.over_under_line = pd.to_numeric(betting.over_under_line)
         betting['over_hit'] = (betting.score_home + betting.score_away > betting.over_under_li
         betting['score_spread'] = betting['score_home']-betting['score_away']
```

```
In [ ]:
```

```
In [76]: print(betting.head())
```

```
     schedule_date  schedule_season schedule_week  schedule_playoff team_home  \
350     1968-01-14             1967     Superbowl              True        GB
538     1969-01-12             1968     Superbowl              True       IND
727     1970-01-11             1969     Superbowl              True        KC
916     1971-01-17             1970     Superbowl              True       IND
1105    1972-01-16             1971     Superbowl              True       DAL

      score_home  score_away team_away team_favorite_id  spread_favorite  \
```

```
350         33.0        14.0       OAK               GB             13.5
538          7.0        16.0       NYJ              IND             18.0
727         23.0         7.0       MIN              MIN             12.0
916         16.0        13.0       DAL              IND              2.5
1105        24.0         3.0       MIA              DAL              6.0

      over_under_line         stadium  stadium_neutral  weather_temperature  \
350              43.0     Orange Bowl             True                 60.0
538              40.0     Orange Bowl             True                 66.0
727              39.0  Tulane Stadium             True                 55.0
916              36.0     Orange Bowl             True                 59.0
1105             34.0  Tulane Stadium             True                 34.0

      weather_wind_mph weather_humidity weather_detail  home_favorite  \
350               12.0               74            NaN           True
538               12.0               80            NaN           True
727               14.0               84            NaN          False
916               11.0               60            NaN           True
1105              18.0               40            NaN           True

      over_hit  score_spread
350       True          19.0
538      False          -9.0
727      False          16.0
916      False           3.0
1105     False          21.0
```

1.1.3 Comparing Over-Under Lines with Spread Coverage

```
In [77]: def covered(x):
             if (x['home_favorite']):
                 if (x['score_spread'] > x['spread_favorite']):
                     return True
                 else:
                     return False
             else:
                 if ((-1*x['score_spread']) > x['spread_favorite']):
                     return True
                 else:
                     return False

         betting['covered_spread'] = betting.apply(lambda x: covered(x), axis = 1)
         betting.head()
```

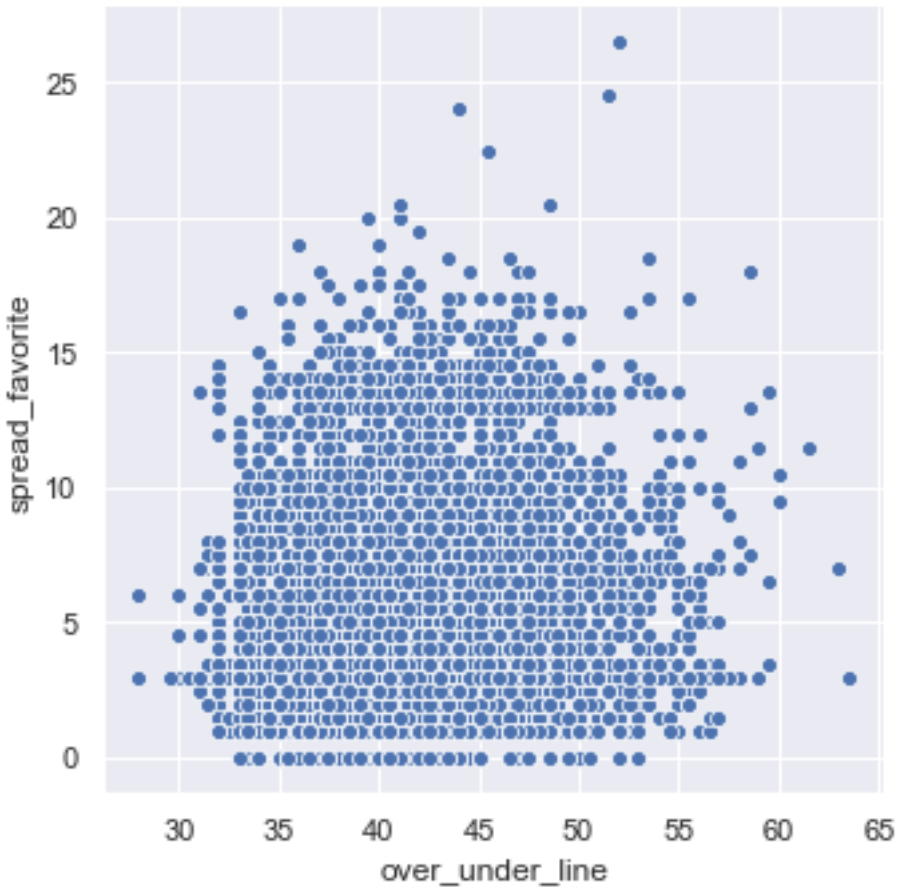
```
Out[77]:      schedule_date  schedule_season schedule_week  schedule_playoff team_home  \
         350     1968-01-14             1967     Superbowl              True        GB
         538     1969-01-12             1968     Superbowl              True       IND
```

```
727     1970-01-11            1969     Superbowl             True        KC
916     1971-01-17            1970     Superbowl             True       IND
1105    1972-01-16            1971     Superbowl             True       DAL

      score_home  score_away team_away team_favorite_id  spread_favorite  ...  \
350         33.0        14.0       OAK              GB             13.5  ...
538          7.0        16.0       NYJ             IND             18.0  ...
727         23.0         7.0       MIN             MIN             12.0  ...
916         16.0        13.0       DAL             IND              2.5  ...
1105        24.0         3.0       MIA             DAL              6.0  ...

              stadium stadium_neutral  weather_temperature  weather_wind_mph  \
350      Orange Bowl            True                 60.0              12.0
538      Orange Bowl            True                 66.0              12.0
727   Tulane Stadium            True                 55.0              14.0
916      Orange Bowl            True                 59.0              11.0
1105  Tulane Stadium            True                 34.0              18.0

      weather_humidity weather_detail home_favorite  over_hit  score_spread  \
350                 74            NaN          True      True          19.0
538                 80            NaN          True     False          -9.0
727                 84            NaN         False     False          16.0
916                 60            NaN          True     False           3.0
1105                40            NaN          True     False          21.0

      covered_spread
350             True
538            False
727            False
916             True
1105            True

[5 rows x 21 columns]
```

```
In [78]: cover = betting.covered_spread.sum()
         total = betting.covered_spread.count()
         percentCoveredSpread = cover / total

         plt.figure(2)
         sns.set(style="darkgrid")
         sns.relplot(x="over_under_line", y ="spread_favorite", data = betting)
```

```
Out[78]: <seaborn.axisgrid.FacetGrid at 0x1a1b5cce10>

<Figure size 432x288 with 0 Axes>
```

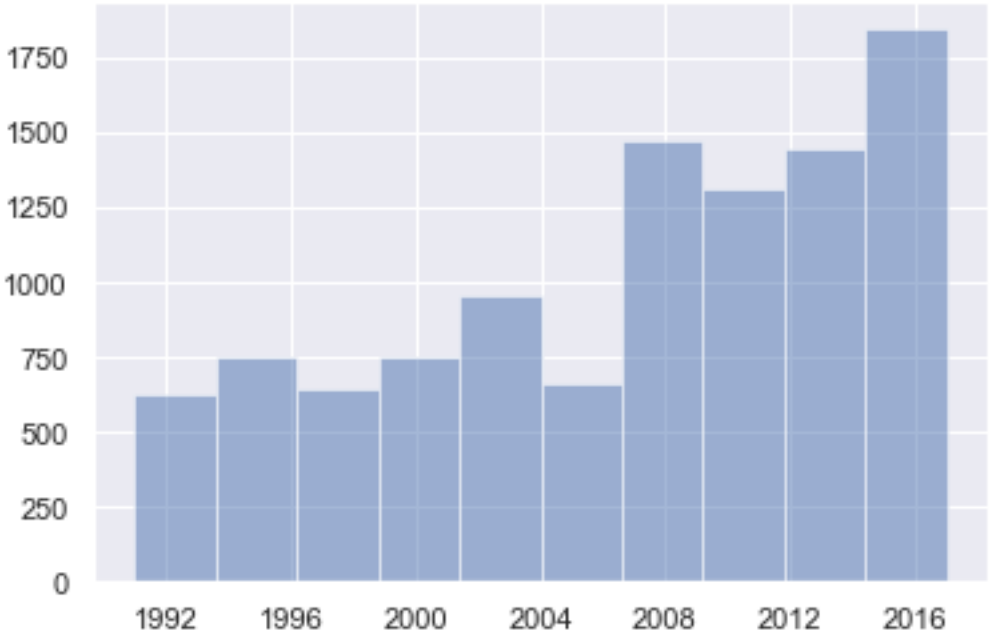

```
In [79]: plt.figure(2)
         sns.set(style="darkgrid")
         sns.relplot(x="over_under_line", y ="spread_favorite", data = betting)

Out[79]: <seaborn.axisgrid.FacetGrid at 0x1a1e7d0208>

<Figure size 432x288 with 0 Axes>
```



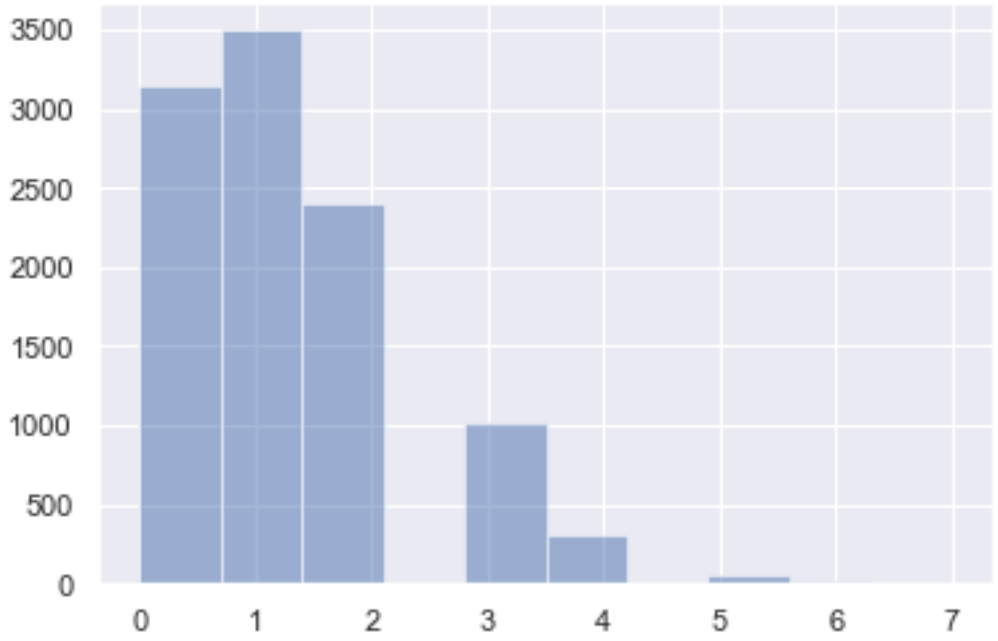

```
In [80]: qb.Year = pd.to_numeric(qb.Year)
         qb.Year.head()

Out[80]: 0    2007
         1    2007
         2    2007
         3    2007
         4    1974
         Name: Year, dtype: int64

In [81]: qb.Year = qb.Year.apply(lambda x: x % 100)
         qb.Year = qb.Year.apply(lambda x: format(x, '02d'))
         qb.Year = qb.Year.apply(lambda x: str(x))
         qb["newDate"] = qb["Game Date"].map(str) + "/" + qb.Year.map(str)
         qb["newDate"] = pd.to_datetime(qb["newDate"])
         qb = qb[qb["Games Started"] == "1"]
```

1.2 Investigating passing data

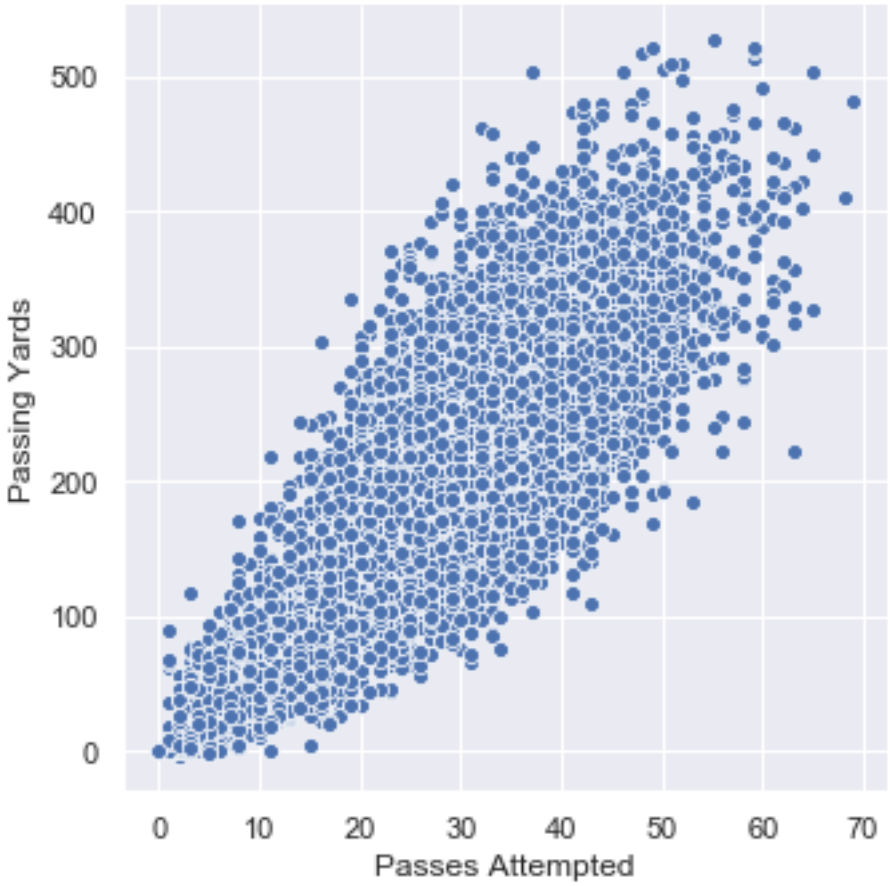
```
In [82]: plt.figure(3)
         plt.hist(qb.newDate, alpha=0.5)

         qb = qb[qb["TD Passes"] != '--']
         qb = qb[qb['Passing Yards'] != '--']
         qb = qb[qb['Passes Completed'] != '--']

         qb['TD Passes'] = pd.to_numeric(qb['TD Passes'])
         qb['Passes Attempted'] = pd.to_numeric(qb['Passes Attempted'])
         qb['Passing Yards'] = pd.to_numeric(qb['Passing Yards'])
```



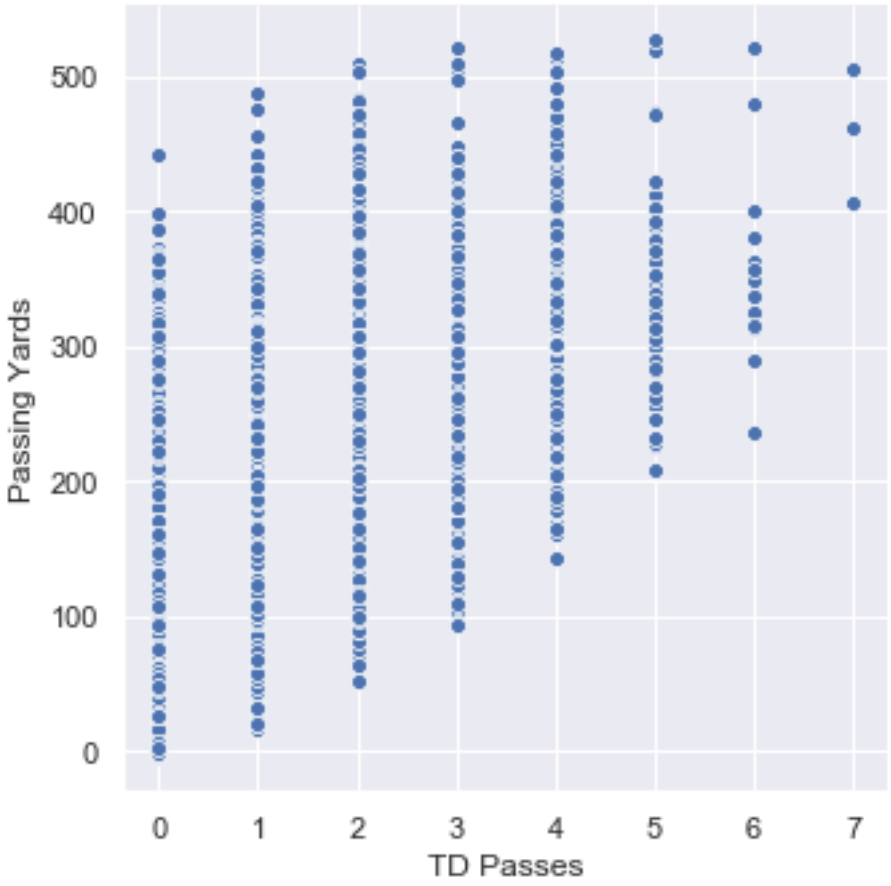

```
In [83]: plt.figure(4)
         plt.hist(qb['TD Passes'], alpha=0.5)

Out[83]: (array([3.142e+03, 3.497e+03, 2.404e+03, 0.000e+00, 1.020e+03, 3.170e+02,
                 0.000e+00, 5.900e+01, 1.700e+01, 3.000e+00]),
          array([0. , 0.7, 1.4, 2.1, 2.8, 3.5, 4.2, 4.9, 5.6, 6.3, 7. ]),
          <a list of 10 Patch objects>)
```




```
In [84]: plt.figure(5)
         sns.relplot(x="Passes Attempted", y ="Passing Yards", data = qb)

Out[84]: <seaborn.axisgrid.FacetGrid at 0x1a1eda7828>

<Figure size 432x288 with 0 Axes>
```




```
In [85]: plt.figure(6)
         sns.relplot(x="TD Passes", y ="Passing Yards", data = qb)

Out[85]: <seaborn.axisgrid.FacetGrid at 0x1a20a6b860>

<Figure size 432x288 with 0 Axes>
```




```
In [86]: final = pd.merge(qb, betting, left_on = ['newDate', 'Opponent'],
                          right_on = ['schedule_date', 'team_away'], how = 'inner')

         final2 = pd.merge(qb, final, left_on = ['newDate', 'Opponent'], right_on = ['schedule
         final2.head()
```

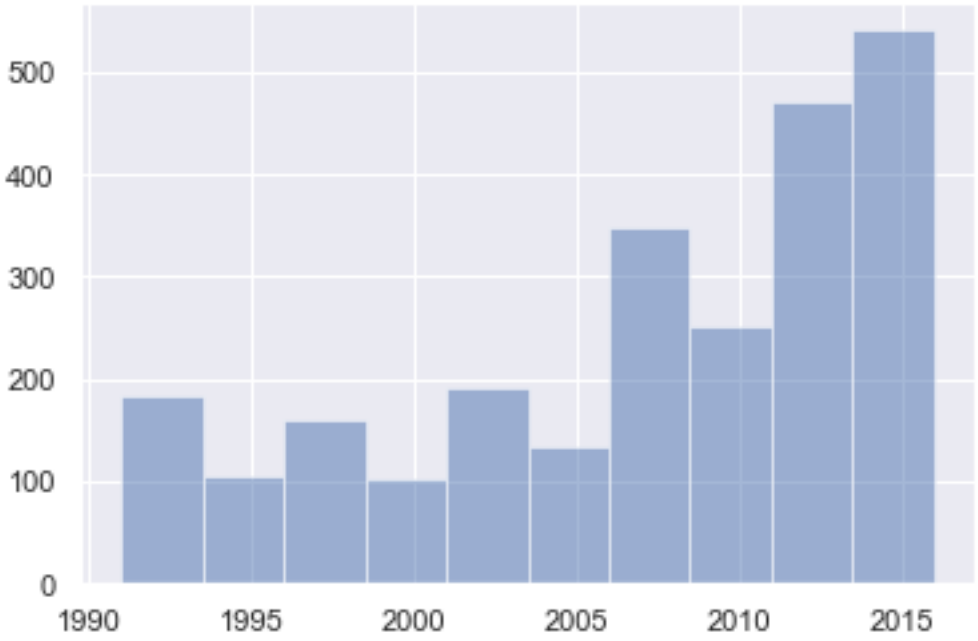
Out[86]:

	Player Id_x	Name_x	Position_x	Year_x	Season_x	\
0	tommymaddox/2501842	Maddox, Tommy	NaN	03	Regular Season	
1	tommymaddox/2501842	Maddox, Tommy	NaN	03	Regular Season	
2	tommymaddox/2501842	Maddox, Tommy	NaN	03	Regular Season	
3	tommymaddox/2501842	Maddox, Tommy	NaN	03	Regular Season	
4	tommymaddox/2501842	Maddox, Tommy	NaN	02	Regular Season	

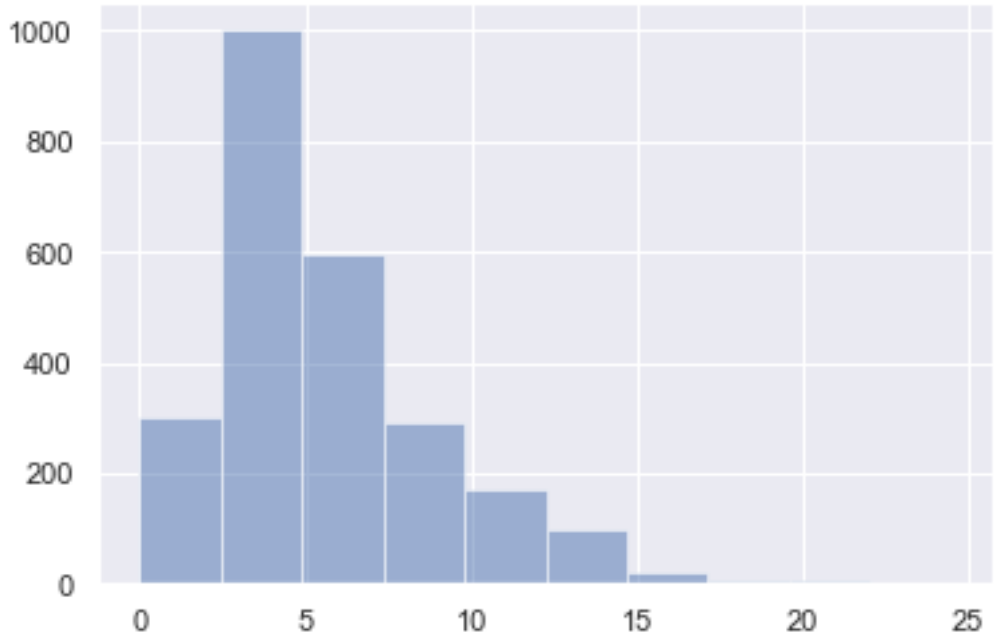
	Week_x	Game Date_x	Home or Away_x	Opponent_x	Outcome_x	...	\
0	3	09/21	Away	CIN	W	...	
1	9	11/02	Away	SEA	L	...	
2	11	11/17	Away	SF	L	...	
3	15	12/14	Away	NYJ	L	...	
4	6	10/13	Away	CIN	W	...	

```
                stadium  stadium_neutral weather_temperature weather_wind_mph  \
0  Paul Brown Stadium            False                63.0              5.0
1   CenturyLink Field            False                38.0              9.0
2    Candlestick Park            False                57.0             14.0
3      Giants Stadium            False                31.0             17.0
4  Paul Brown Stadium            False                58.0             12.0

   weather_humidity weather_detail  home_favorite over_hit  score_spread  \
0                73            NaN          False    False          -7.0
1                81            NaN           True    False           7.0
2                86            NaN           True     True          16.0
3                65            NaN           True    False           6.0
4                76            NaN          False     True         -27.0

  covered_spread
0           True
1           True
2           True
3           True
4           True

[5 rows x 81 columns]
```

In [87]:
```
plt.figure(7)
plt.hist(final2['schedule_season'], alpha=0.5)

df = final2.drop(columns  = ["Position_x", "Year_x", "Game Date_x", "Games Played_x",
                             "Position_y", "Year_y", "Game Date_y", "Games Played_
```



In [88]:
```
plt.figure(8)
plt.hist(df['spread_favorite'], alpha=0.5)

def FavoriteQBYards(x):
    if (x['home_favorite']):
        return x['Passing Yards_y']
    else:
        return x['Passing Yards_x']

df['favorite_passing_yards'] = df.apply(lambda x: FavoriteQBYards(x), axis = 1)
```

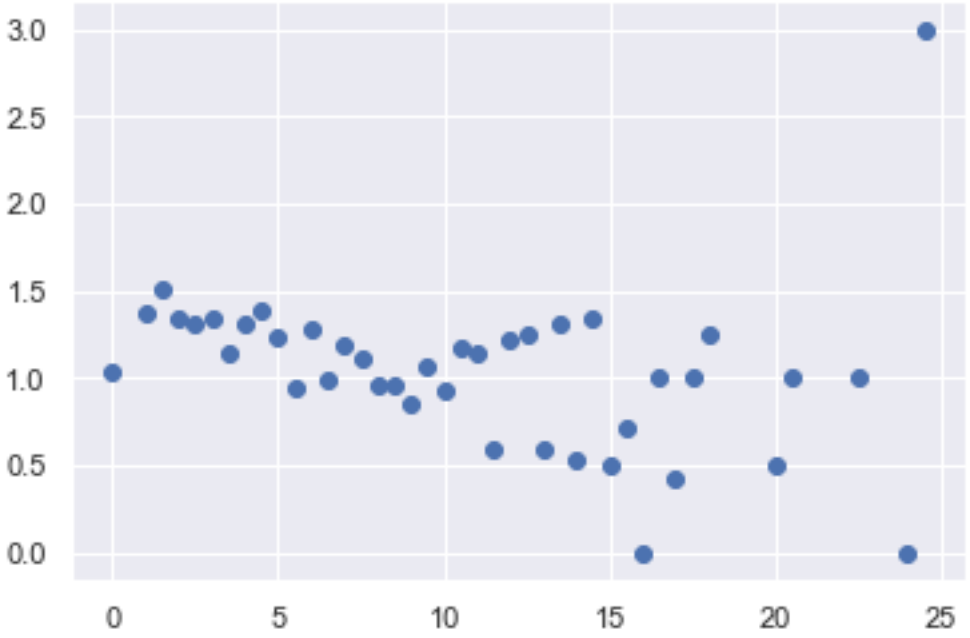


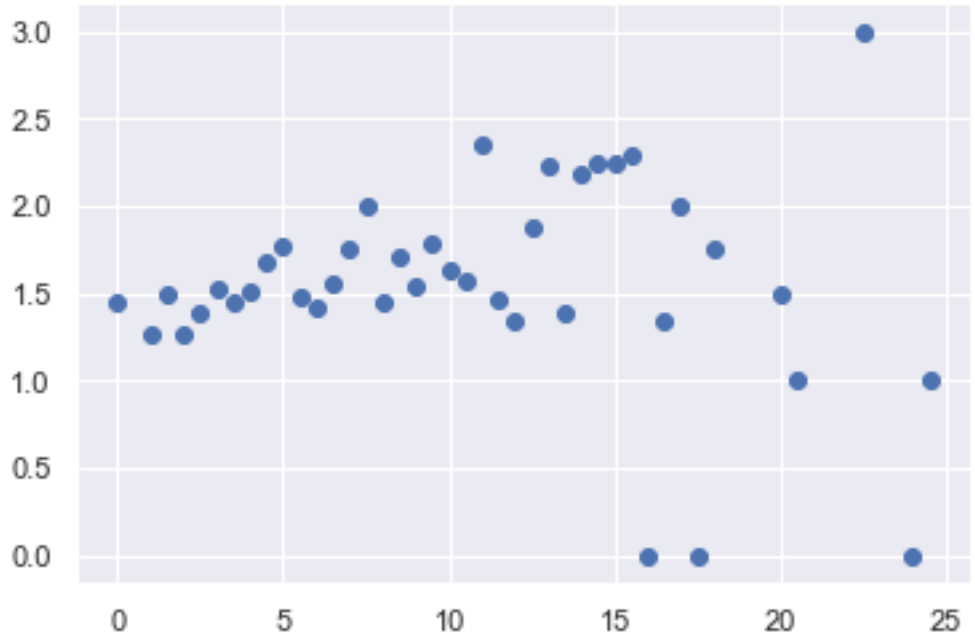
1.3 Comparing Passing Data with Spread Coverage

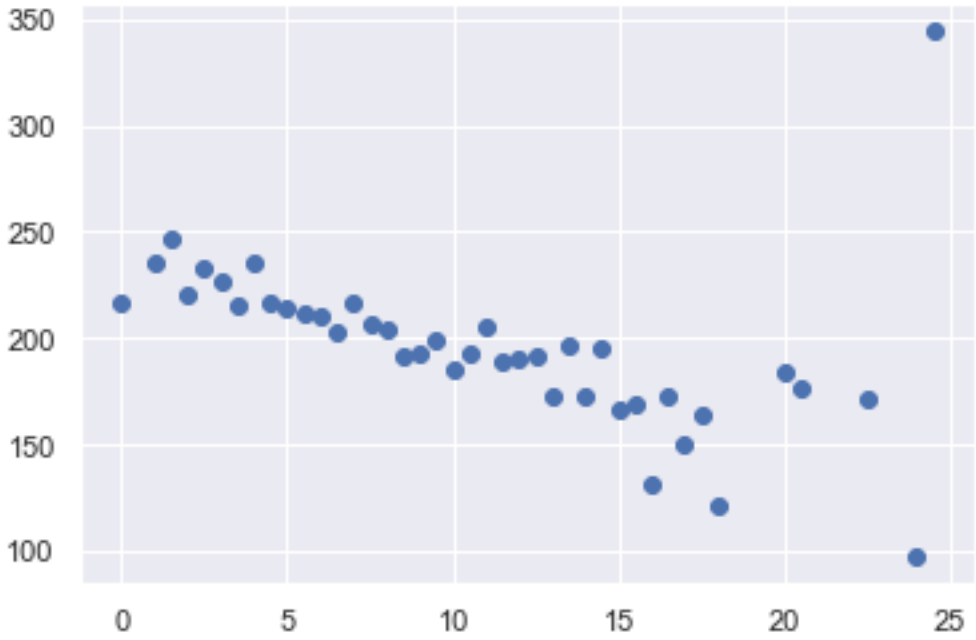
```
In [89]: def UnderdogQBYards(x):
             if (x['home_favorite']):
                 return x['Passing Yards_x']
             else:
                 return x['Passing Yards_y']

         df['underdog_passing_yards'] = df.apply(lambda x: UnderdogQBYards(x), axis = 1)

         def FavoriteQBTD(x):
             if (x['home_favorite']):
                 return x['TD Passes_y']
             else:
                 return x['TD Passes_x']

         df['favorite_TD_passes'] = df.apply(lambda x: FavoriteQBTD(x), axis = 1)

         def UnderdogQBTD(x):
             if (x['home_favorite']):
                 return x['TD Passes_x']
             else:
                 return x['TD Passes_y']

         df['underdog_TD_passes'] = df.apply(lambda x: UnderdogQBTD(x), axis = 1)
```

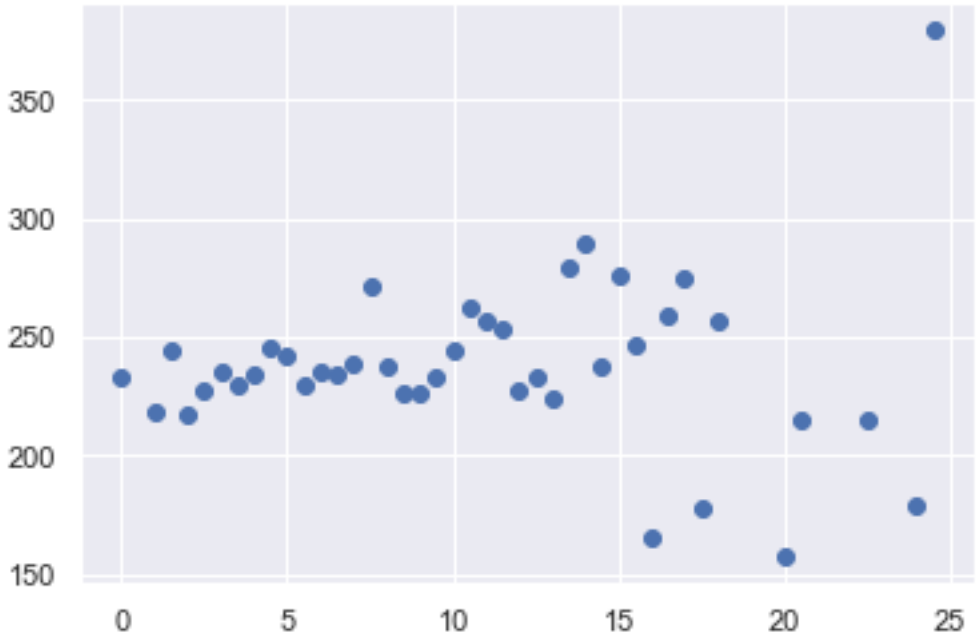
In [90]:
```
plt.figure(9)
a = df.groupby(df['spread_favorite'])['underdog_TD_passes'].mean()
plt.scatter(a.index, a)

plt.figure(10)
a = df.groupby(df['spread_favorite'])['favorite_TD_passes'].mean()
plt.scatter(a.index, a)

plt.figure(11)
a = df.groupby(df['spread_favorite'])['underdog_passing_yards'].mean()
plt.scatter(a.index, a)

plt.figure(12)
a = df.groupby(df['spread_favorite'])['favorite_passing_yards'].mean()
plt.scatter(a.index, a)
```

Out[90]:
```
<matplotlib.collections.PathCollection at 0x1a1da5cfd0>
```




3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25


350
300
250
200
150
100
0
5
10
15
20
25



1.4 Training models using ensemble methods

```
In [91]: reduced_df = df[['underdog_TD_passes', 'favorite_TD_passes', 'underdog_passing_yards'
                          'favorite_passing_yards', 'home_favorite', 'team_home', 'team_away'
                          'score_home', 'score_away', 'team_away', 'team_favorite_id', 'sprea
                          'over_under_line', 'score_spread', 'covered_spread', 'over_hit']]

         train_spread = reduced_df[['underdog_TD_passes', 'favorite_TD_passes', 'underdog_pass
                                    'favorite_passing_yards', 'spread_favorite', 'covered_sprea



         train_over = reduced_df[['underdog_TD_passes', 'favorite_TD_passes', 'underdog_passing
                                  'favorite_passing_yards', 'over_under_line', 'over_hit']]

In [92]: X_spread = train_spread.drop('covered_spread', axis=1)
         y_spread = train_spread['covered_spread']
         X_over = train_over.drop('over_hit',axis=1)
         y_over = train_over['over_hit']

In [93]: base = LDA()
         # choose 7 best features
         rfe = RFE(base, 5)
         rfe = rfe.fit(X_over, y_over)

In [94]: # features
         print(rfe.support_)
```

```
        print(rfe.ranking_)

        boost = xgb.XGBClassifier()
        X_train_over, X_test_over, y_train_over, y_test_over = train_test_split(
            X_over, y_over, test_size=0.33, random_state=42)
        boost.fit(X_train_over, y_train_over)
```

```
[ True  True  True  True  True]
[1 1 1 1 1]
```

```
Out[94]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                       colsample_bynode=1, colsample_bytree=1, gamma=0,
                       learning_rate=0.1, max_delta_step=0, max_depth=3,
                       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                       nthread=None, objective='binary:logistic', random_state=0,
                       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                       silent=None, subsample=1, verbosity=1)
```

```
In [95]: preds = boost.predict(X_test_over)
         print(roc_auc_score(y_test_over, preds))
         print(accuracy_score(y_test_over, preds, normalize=True))
         models = []

         models.append(('LRG', LogisticRegression(solver='liblinear')))
         models.append(('KNB', KNeighborsClassifier()))
         models.append(('GNB', GaussianNB()))
         models.append(('XGB', xgb.XGBClassifier(random_state=0)))
         models.append(('RFC', RandomForestClassifier(random_state=0, n_estimators=100)))
         models.append(('DTC', DecisionTreeClassifier(random_state=0, criterion='entropy', max_
```

```
0.7465807812648146
0.7445255474452555
```

```
In [96]: # evaluate each model by average and standard deviations of roc auc
         results = []
         names = []

         for name, m in models:
             kfold = model_selection.KFold(n_splits=5, random_state=0)
             cv_results = model_selection.cross_val_score(m, X_over, y_over, cv=kfold, scoring
             results.append(cv_results)
             names.append(name)
             msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
             print(msg)


         boost = xgb.XGBClassifier()
```

```
        rfc = RandomForestClassifier(random_state=0, n_estimators=100)
        lrg = LogisticRegression(solver='liblinear')
        vote = VotingClassifier(estimators=[('boost', boost), ('rfc', rfc), ('lrg', lrg)], vo

        model = CCV(vote, method='isotonic', cv=3)
        model.fit(X_train_over, y_train_over)
        preds = model.predict(X_test_over)
        print(accuracy_score(y_test_over, preds, normalize=True))
```

```
LRG: 0.820192 (0.020541)
KNB: 0.627767 (0.013754)
GNB: 0.786904 (0.024047)
XGB: 0.816006 (0.015937)
RFC: 0.781894 (0.014028)
DTC: 0.784632 (0.016691)
0.745742092457421
```

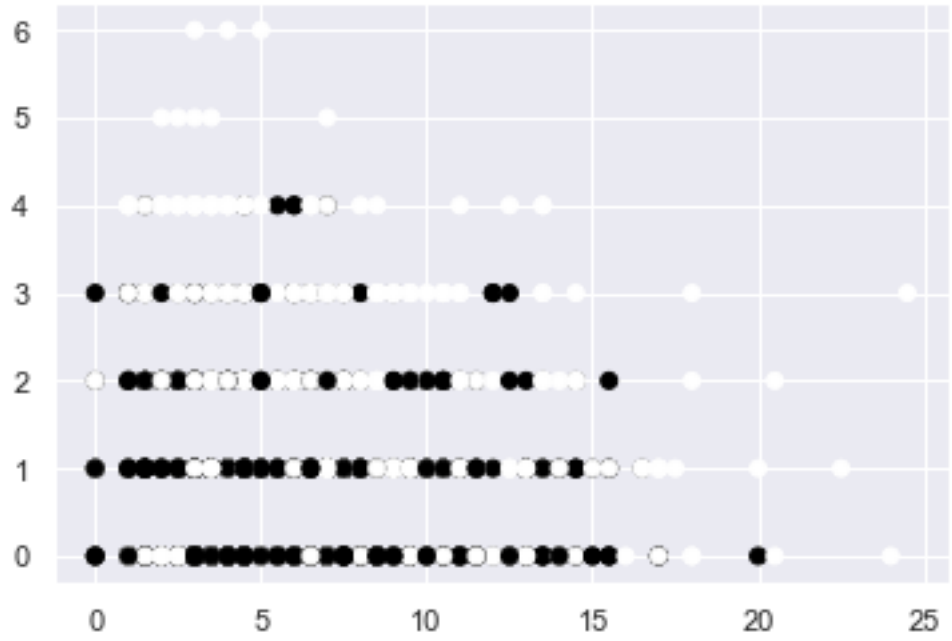
```
In [97]: rfe = RFE(base, 5)
         rfe = rfe.fit(X_spread, y_spread)

         # features
         print(rfe.support_)
         print(rfe.ranking_)

         boost = xgb.XGBClassifier()
         X_train_spread, X_test_spread, y_train_spread, y_test_spread = train_test_split(
             X_spread, y_spread, test_size=0.33, random_state=42)
         boost.fit(X_train_spread, y_train_spread)

         preds = boost.predict(X_test_spread)

         print(roc_auc_score(y_test_spread, preds))

         print(accuracy_score(y_test_spread, preds, normalize=True))

         models = []

         models.append(('LRG', LogisticRegression(solver='liblinear')))
         models.append(('KNB', KNeighborsClassifier()))
         models.append(('GNB', GaussianNB()))
         models.append(('XGB', xgb.XGBClassifier(random_state=0)))
         models.append(('RFC', RandomForestClassifier(random_state=0, n_estimators=100)))
         models.append(('DTC', DecisionTreeClassifier(random_state=0, criterion='entropy', max_

         # evaluate each model by average and standard deviations of roc auc
         results = []
         names = []
```

```
        for name, m in models:
            kfold = model_selection.KFold(n_splits=5, random_state=0)
            cv_results = model_selection.cross_val_score(m, X_spread, y_spread, cv=kfold, sco
            results.append(cv_results)
            names.append(name)
            msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
            print(msg)
```

```
[ True  True  True  True  True]
[1 1 1 1 1]
0.68708975456621
0.6909975669099757
LRG: 0.732243 (0.013634)
KNB: 0.536384 (0.018879)
GNB: 0.717054 (0.011590)
XGB: 0.720706 (0.012391)
RFC: 0.682868 (0.016307)
DTC: 0.672994 (0.024256)
```

```
In [98]: boost = xgb.XGBClassifier()
         rfc = RandomForestClassifier(random_state=0, n_estimators=100)
         lrg = LogisticRegression(solver='liblinear')
         vote = VotingClassifier(estimators=[('boost', boost), ('rfc', rfc), ('lrg', lrg)], vo

         model = CCV(vote, method='isotonic', cv=3)
         model.fit(X_train_spread, y_train_spread)
         preds = model.predict(X_test_spread)
         print(accuracy_score(y_test_spread, preds, normalize=True))


         fig, ax = plt.subplots()
         scatter = ax.scatter(reduced_df.spread_favorite, reduced_df.underdog_TD_passes,
                              c = reduced_df.covered_spread, cmap="binary")
```

```
0.6982968369829684
```

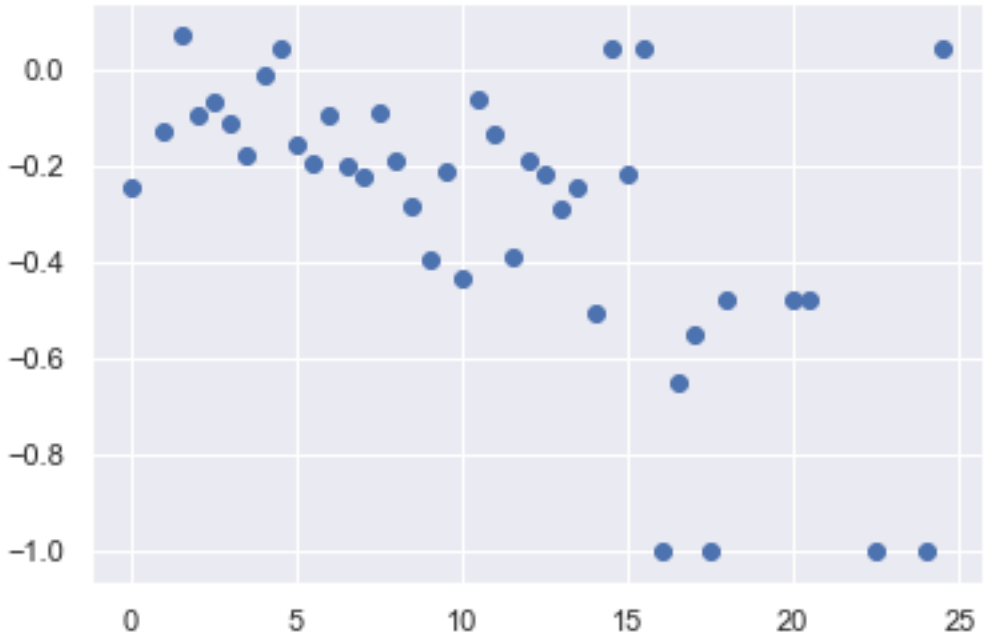


1.5 Trying to find Arbitrage

```
In [99]: ranger = reduced_df.spread_favorite.unique()

         EV_total = np.zeros(ranger.size)
         j = 0
         for i in ranger:

             grouped  = reduced_df.groupby('spread_favorite')
             spread = i

             # td_passes = grouped.get_group(spread)['underdog_TD_passes'].mean()
             td_passes = 1.5

             current  = grouped.get_group(spread)
             size = current.shape[0]
             covered_set = current[current['covered_spread']]
             size_covered_set = covered_set.shape[0]
             size_both = covered_set[covered_set['underdog_TD_passes'] > td_passes].shape[0]
             underdog_overTD = current[current['underdog_TD_passes'] > td_passes]
             size_underdog_overTD = underdog_overTD.shape[0]
             sanity_check_size = underdog_overTD[underdog_overTD['covered_spread']].shape[0]

             percent_covered = size_covered_set/ size
             percent_underTD = size_underdog_overTD/ size
```

```
            # print('Covered Spread:', end = ' ')
            # print(percent_covered)
            # print('Under', end = ' ')
            # print(td_passes, end  = '')
            # print(':', end = ' ')
            # print(percent_underTD)


            # if(percent_covered <= 50):
            #     m1 = 100*(1-percent_covered) / percent_covered
            #     a1 = (100 + m1) / 100
            # else:
            #     m1 = 100*(percent_covered) / (1-percent_covered)
            #     a1 = (100 + m1) / m1

            # if(percent_underTD <= 50):
            #     m2 = 100*(1-percent_underTD) / percent_underTD
            #     a2 = (100 + m2) / 100
            # else:
            #     m2 = 100*(percent_underTD) / (1-percent_underTD)
            #     a2 = (100 + m2) / m2

            a1 = 210/110
            a2 = 230/100
            f1 = a2 / (a1 + a2)
            f2 = a1 / (a2 + a1)

            if (sanity_check_size != size_both):
                print(spread)
                print('error')

            alpha = size_both / size
            beta = (size_covered_set - size_both) / size
            gamma = (size_underdog_overTD - size_both) / size

            EV = alpha * (a1 * f1 + a2*f2) + beta * (a1 * f1) + gamma * (a2 * f2) - f1 - f2

            EV_total[j] = EV
            j = j+ 1

        EV_graph = pd.DataFrame(ranger, EV_total)

        plt.scatter(EV_graph, EV_graph.index)
```

Out[99]: <matplotlib.collections.PathCollection at 0x1a1edb4b38>



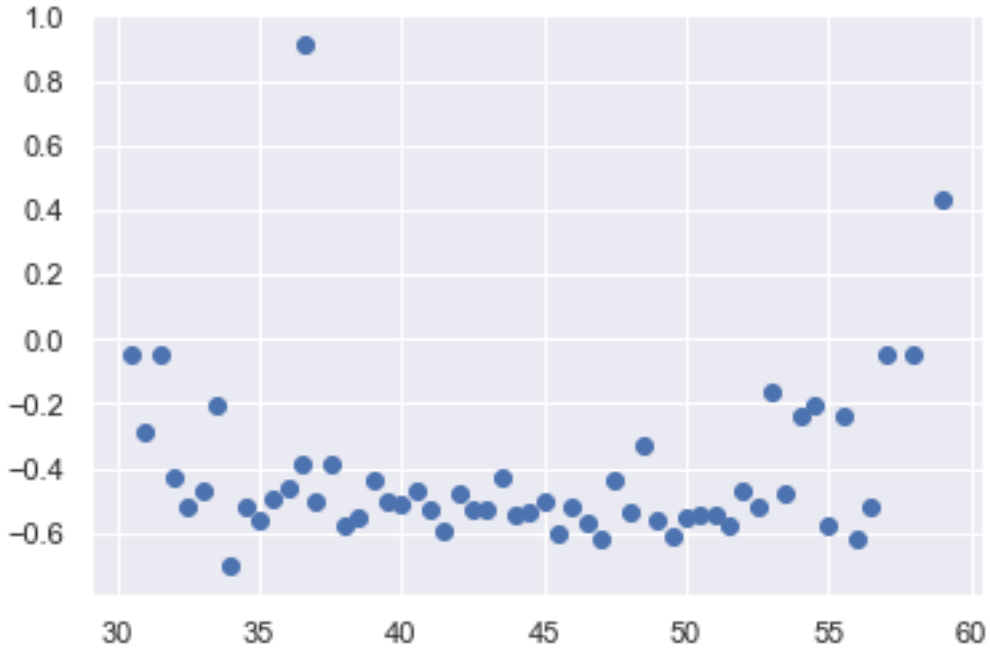
In [100]:
```
ranger = reduced_df.over_under_line.unique()

EV_total = np.zeros(ranger.size)
j = 0
for i in ranger:

    grouped  = reduced_df.groupby('over_under_line')
    over = i

    # td_passes = grouped.get_group(spread)['underdog_TD_passes'].mean()
    td_passes = 1.5

    current  = grouped.get_group(over)
    size = current.shape[0]
    covered_set = current[current['over_hit']]
    size_covered_set = covered_set.shape[0]
    size_both = covered_set[covered_set['over_hit'] > td_passes].shape[0]
    underdog_overTD = current[current['over_hit'] > td_passes]
    size_underdog_underTD = underdog_overTD.shape[0]
    sanity_check_size = underdog_overTD[underdog_underTD['over_hit']].shape[0]

    percent_covered = size_covered_set/ size
    percent_overTD = size_underdog_overTD/ size

    a1 = 210/110
```

```
        a2 = 210/110
        f1 = a2 / (a1 + a2)
        f2 = a1 / (a2 + a1)

        if (sanity_check_size != size_both):
            print(spread)
            print('error')

        alpha = size_both / size
        beta = (size_covered_set - size_both) / size
        gamma = (size_underdog_overTD - size_both) / size

        EV = alpha * (a1 * f1 + a2*f2) + beta * (a1 * f1) + gamma * (a2 * f2) - f1 - f2

        EV_total[j] = EV
        j = j+ 1

    EV_graph = pd.DataFrame(ranger, EV_total)

    plt.scatter(EV_graph, EV_graph.index)
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series ke
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:20: UserWarning: Boolean Series k
```

```
Out[100]: <matplotlib.collections.PathCollection at 0x1a1ed79ba8>
```


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
30
35
40
45
50
55
60