elo_model

March 22, 2024

[ ]:

```
import pandas as pd
import numpy as np

import clean_dataset
import optuna
```

[ ]:

```
all_df = clean_dataset.get_dataset()
mapping = clean_dataset.get_mapping()
all_df = clean_dataset.create_cols(all_df, mapping)
```

[ ]:

```
team_names = list(mapping.values())
assert len(team_names) == 32
```

[ ]:

```
# returns a train and test set copy of a given dataframe
def top_percent(df, percent):
    num_rows = df.shape[0]
    cutoff = int(num_rows*percent)
    return df.iloc[0:cutoff, :].copy(deep=True), df.iloc[cutoff:, :].
 ↪copy(deep=True)
```

[ ]:

```
df_2024 = all_df.query("season == '2023-24 season:'")
df_2023 = all_df.query("season == '2022-23 season:'")
```

[ ]:

```
train_2024, val_df = top_percent(df_2024, 0.6)
train_df = pd.concat([df_2023, train_2024])

print(f"train size = {train_df.shape[0]}")
print(f"val size = {val_df.shape}")
```

```
train size = 1742
val size = (287, 14)
```

[ ]:

```
from sklearn.base import BaseEstimator
from sklearn import metrics
from copy import deepcopy

class EloClassifier(BaseEstimator):
    def __init__(self, team_names, initial_elo, k, alpha, home_adv,
                 mov_exp, mov_bias, auto_coeff, auto_bias, k_mult=None,
                 use_draw=False, elo_scores_dict=dict()):
        self.team_names = team_names
        self.initial_elo = initial_elo
        self.k = k
        self.alpha = alpha
        self.home_adv = home_adv
        self.mov_exp = mov_exp
        self.mov_bias = mov_bias
        self.auto_coeff = auto_coeff
        self.auto_bias = auto_bias
        self.elo_scores_dict = elo_scores_dict
        self.k_mult = k_mult
        self.use_draw = use_draw

    def create_initial_elo(self):
        keys = self.team_names
        values = self.initial_elo*np.ones(len(keys))
        self.elo_scores_dict = dict(zip(keys, values))

    def win_prob(self, team_rating: float, opponent_rating: float) -> float:
        return 1/(1 + np.power(10, -(team_rating - opponent_rating)/400))   # 1/
↪(1 + 10^(-diff/400))

    # returns sorted list of all team rankings
    def return_sorted_list(self):
        return [(k,v) for k, v in sorted(self.elo_scores_dict.items(),␣
↪key=lambda item: item[1], reverse=True)]

    def fit(self, df, y=None):

        self.create_initial_elo()
        current_season = df.iloc[0,:]['numeric_season']
        k_multiplier=1

        # iterate over every row of dataframe
        for row_index in range(len(df)):
            game_row = df.iloc[row_index, :]    # pandas series

            # check if season change
            if current_season < game_row['numeric_season']:
                self.elo_scores_dict.update( (k, self.alpha*v + (1-self.
↪alpha)*self.initial_elo) for k, v in self.elo_scores_dict.items() )
                current_season = game_row['numeric_season']

            # get elo values of teams
            home_original_elo = self.elo_scores_dict[game_row['home_team']]
            away_original_elo = self.elo_scores_dict[game_row['away_team']]

            # calculate expected scores (probs)
            home_prob = self.win_prob(home_original_elo + self.home_adv,␣
↪away_original_elo)
            away_prob = self.win_prob(away_original_elo, home_original_elo +␣
↪self.home_adv)

            # Calculate new elo values
            if self.k_mult == '538':
                elo_diff = home_original_elo + self.home_adv - away_original_elo
                mov = np.abs(game_row['home_score'] - game_row['away_score'])
                k_multiplier = np.power(mov + self.mov_bias, self.mov_exp)/
↪(self.auto_bias + self.auto_coeff*elo_diff)
            elif self.k_mult == '538_specific':
                elo_diff = home_original_elo + self.home_adv - away_original_elo
                mov = np.abs(game_row['home_score'] - game_row['away_score'])
                k_multiplier = (0.6686 * np.log(mov) + 0.8048) * (2.05 /␣
↪(elo_diff * 0.001 + 2.05))
            elif self.k_mult == 'multiplicative':
                elo_diff = home_original_elo + self.home_adv - away_original_elo
                mov = np.abs(game_row['home_score'] - game_row['away_score'])/
↪self.mov_bias
                k_multiplier = np.power( 1 + mov, self.mov_exp)
            else:
                k_multiplier = 1

            if self.use_draw and game_row['overtime']:  # treat it as a draw
                home_updated_elo = home_original_elo + self.k*k_multiplier*(0.5␣
↪- home_prob)
                away_updated_elo = away_original_elo + self.k*k_multiplier*(0.5␣
↪- away_prob)
            else:
                home_updated_elo = home_original_elo + self.
↪k*k_multiplier*(game_row['home_win'] - home_prob)
                away_updated_elo = away_original_elo + self.
↪k*k_multiplier*(game_row['away_win'] - away_prob)

            # update ELO values
            self.elo_scores_dict[game_row['home_team']] = home_updated_elo
            self.elo_scores_dict[game_row['away_team']] = away_updated_elo

    # update_ratings: boolean if you want the elo scores to update
    # after the seen validation sample or not
    def predict_proba(self, df, update_ratings=True):
        current_season = df.iloc[0,:]['numeric_season']
        elo_dict = deepcopy(self.elo_scores_dict)
        probabilities = []
        k_multiplier=1

        # iterate over every row of dataframe
        for row_index in range(len(df)):
            game_row = df.iloc[row_index, :]    # pandas series

            # check if season change
            if current_season < game_row['numeric_season']:
                elo_dict.update( (k, self.alpha*v + (1-self.alpha)*self.
↪initial_elo) for k, v in elo_dict.items() )
                current_season = game_row['numeric_season']

            # get elo values of teams
            home_original_elo = elo_dict[game_row['home_team']]
            away_original_elo = elo_dict[game_row['away_team']]
            # calculate expected scores (probs)
            home_prob = self.win_prob(home_original_elo + self.home_adv,␣
↪away_original_elo)
            away_prob = self.win_prob(away_original_elo, home_original_elo +␣
↪self.home_adv)
            probabilities.append([home_prob, away_prob])

            if update_ratings:
                # Calculate new elo values
                if self.k_mult == '538':
                    elo_diff = home_original_elo + self.home_adv -␣
↪away_original_elo
                    mov = np.abs(game_row['home_score'] -␣
↪game_row['away_score'])
                    k_multiplier = np.power(mov + self.mov_bias, self.mov_exp)/
↪(self.auto_bias + self.auto_coeff*elo_diff)
                elif self.k_mult == '538_specific':
                    elo_diff = home_original_elo + self.home_adv -␣
↪away_original_elo
                    mov = np.abs(game_row['home_score'] -␣
↪game_row['away_score'])
                    k_multiplier = (0.6686 * np.log(mov) + 0.8048) * (2.05/
↪(elo_diff * 0.001 + 2.05))
                elif self.k_mult == 'multiplicative':
                    elo_diff = home_original_elo + self.home_adv -␣
↪away_original_elo
                    mov = np.abs(game_row['home_score'] -␣
↪game_row['away_score'])/self.mov_bias
                    k_multiplier = np.power( 1 + mov, self.mov_exp)
                else:
                    k_multiplier = 1

                if self.use_draw and game_row['overtime']:  # treat it as a draw
                    home_updated_elo = home_original_elo + self.
 ↪k*k_multiplier*(0.5 - home_prob)
                    away_updated_elo = away_original_elo + self.
 ↪k*k_multiplier*(0.5 - away_prob)
                else:
                    home_updated_elo = home_original_elo + self.
 ↪k*k_multiplier*(game_row['home_win'] - home_prob)
                    away_updated_elo = away_original_elo + self.
 ↪k*k_multiplier*(game_row['away_win'] - away_prob)

                    # update ELO values
                    self.elo_scores_dict[game_row['home_team']] =␣
 ↪home_updated_elo
                    self.elo_scores_dict[game_row['away_team']] =␣
 ↪away_updated_elo

        return np.array(probabilities)

    def predict(self, X, update_ratings=True):
        probabilities = self.predict_proba(X, update_ratings)
        return np.argmax(probabilities, axis=1)

    def score(self, X, y, update_ratings=True):
        probabilities = self.predict_proba(X, update_ratings)
        return metrics.log_loss(y, probabilities[:, 0]) # take probabilities of␣
 ↪home team
```

[ ]:

```
optuna.logging.set_verbosity(optuna.logging.WARNING)

def objective(trial):
    initial_elo = 1500
    ALPHA = trial.suggest_float('alpha', 0.0, 1.0, step=0.1)
    K = trial.suggest_int("k", 0, 40, step=1)
    HOME_ADV = trial.suggest_int("home_adv", 0, 50, step=1)
    MOV_EXP = trial.suggest_float('mov_exp', 0.0, 3.0, step=0.1)
    MOV_BIAS = trial.suggest_float('mov_bias', 0.0, 5.0, step=0.1)
    AUTO_COEFF = trial.suggest_float('auto_coeff', 0.0, 0.1, step=0.001)
    AUTO_BIAS = trial.suggest_float('auto_bias', 1.0, 10.0, step=0.1)
    # MOV_EXP = 1
    # MOV_BIAS = 1
    # AUTO_COEFF = 1
    # AUTO_BIAS = 1
    elo_model = EloClassifier(team_names, initial_elo, K, ALPHA, HOME_ADV,␣
  ↪MOV_EXP, MOV_BIAS, AUTO_COEFF, AUTO_BIAS, k_mult=True, use_draw=True)
    elo_model.fit(train_df)
    return elo_model.score(val_df, val_df[['home_win']].values,␣
  ↪update_ratings=True)

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=100)
print(study.best_trial)
```

```
FrozenTrial(number=92, state=TrialState.COMPLETE, values=[0.6684915705932248],
datetime_start=datetime.datetime(2024, 3, 21, 17, 10, 55, 609955),
datetime_complete=datetime.datetime(2024, 3, 21, 17, 10, 55, 954543),
params={'alpha': 1.0, 'k': 7, 'home_adv': 31, 'mov_exp': 0.5, 'mov_bias':
2.4000000000000004, 'auto_coeff': 0.089, 'auto_bias': 2.9000000000000004},
user_attrs={}, system_attrs={}, intermediate_values={}, distributions={'alpha':
FloatDistribution(high=1.0, log=False, low=0.0, step=0.1), 'k':
IntDistribution(high=40, log=False, low=0, step=1), 'home_adv':
IntDistribution(high=50, log=False, low=0, step=1), 'mov_exp':
FloatDistribution(high=3.0, log=False, low=0.0, step=0.1), 'mov_bias':
FloatDistribution(high=5.0, log=False, low=0.0, step=0.1), 'auto_coeff':
FloatDistribution(high=0.1, log=False, low=0.0, step=0.001), 'auto_bias':
FloatDistribution(high=10.0, log=False, low=1.0, step=0.1)}, trial_id=92,
value=None)
```

[ ]:

```
home_win_percentage = all_df.home_win.sum()/all_df.shape[0]
print(f"league average home team win percentage = {np.
  ↪round(home_win_percentage, 3)}")
```

```
league average home team win percentage = 0.529
```

[ ]:

```
# store information about a specific elo model, as fittd by given␣
  ↪hyperparameters
def diagnostics(model_str, results_df, pred_prob_df, train_df, val_df, K,␣
  ↪ALPHA, HOME_ADV, MOV_EXP=1, MOV_BIAS=1, AUTO_COEFF=1, AUTO_BIAS=1,␣
  ↪K_MULTI=False, USE_DRAW=False, home_win_percentage=0.529):
    home_win_percentage = all_df.home_win.sum()/all_df.shape[0]

    elo_model = EloClassifier(team_names, 1500, k=K, alpha=ALPHA,␣
  ↪home_adv=HOME_ADV, mov_exp=MOV_EXP, mov_bias=MOV_BIAS,␣
  ↪auto_coeff=AUTO_COEFF, auto_bias=AUTO_BIAS, k_mult=K_MULTI,␣
  ↪use_draw=USE_DRAW)
    elo_model.fit(train_df)
    log_loss = elo_model.score(val_df, val_df[['home_win']].values,␣
  ↪update_ratings=True)
    predicted_home_win_prob = elo_model.win_prob(1500+HOME_ADV,1500)
    top_eight_rankings = elo_model.return_sorted_list()[0:8]
    bottom_four_rankings = elo_model.return_sorted_list()[-4:]
    results_df.loc[model_str] = [log_loss, np.round(predicted_home_win_prob -␣
  ↪home_win_percentage,3)] + top_eight_rankings + bottom_four_rankings
    pred_prob_df[f"home_prob_{model_str}"] = elo_model.predict_proba(val_df)[:
  ↪,0]
```

[ ]:

```
results_df = pd.DataFrame(columns=['log_loss', 'home_win_prob_diff', '1', '2',␣
  ↪'3', '4', '5', '6', '7', '8', '-4', '-3', '-2', '-1'])
predicted_probs_df = deepcopy(val_df.loc[:,['date','home_team', 'away_team',␣
  ↪'score_diff']])

diagnostics("538_params", results_df, predicted_probs_df, train_df, val_df, 6,␣
  ↪0.7, 50, MOV_EXP=0, MOV_BIAS=0, AUTO_COEFF=0, AUTO_BIAS=0,␣
  ↪K_MULTI='538_specific')
diagnostics("no_k_multi", results_df, predicted_probs_df, train_df, val_df, 6,␣
  ↪1.0, 32, MOV_EXP=1, MOV_BIAS=1, AUTO_COEFF=1, AUTO_BIAS=1, K_MULTI=False)
diagnostics("no_k_multi_use_draw", results_df, predicted_probs_df, train_df,␣
  ↪val_df, 7, 1.0, 27, MOV_EXP=1, MOV_BIAS=1, AUTO_COEFF=1, AUTO_BIAS=1,␣
  ↪K_MULTI=False, USE_DRAW=True)
diagnostics("k_multi_no_auto_coeff", results_df, predicted_probs_df, train_df,␣
  ↪val_df, 3, 1.0, 28, MOV_EXP=0.7, MOV_BIAS=0.9, AUTO_COEFF=0.0, AUTO_BIAS=1.
  ↪1, K_MULTI='538')
diagnostics("k_multi_no_auto_coeff_use_draw", results_df, predicted_probs_df,␣
  ↪train_df, val_df, 3, 1.0, 28, MOV_EXP=0.7, MOV_BIAS=0.9, AUTO_COEFF=0.0,␣
  ↪AUTO_BIAS=1.1, K_MULTI='538', USE_DRAW=True)
diagnostics("k_multi_small_auto_coeff_use_draw", results_df,␣
  ↪predicted_probs_df, train_df, val_df, 12, 0.8, 28, MOV_EXP=1.5, MOV_BIAS=0.
  ↪3, AUTO_COEFF=0.018, AUTO_BIAS=9.2, K_MULTI='538', USE_DRAW=True)
diagnostics("k_multi_multiplicative", results_df, predicted_probs_df, train_df,␣
  ↪val_df, 29, 1.0, 47, MOV_EXP=0.4, MOV_BIAS=3.3, AUTO_COEFF=1, AUTO_BIAS=1,␣
  ↪K_MULTI='multiplicative')
```

[ ]:

```
results_df
```

[ ]:

```
                                    log_loss  home_win_prob_diff  \
538_params                          0.667089               0.043
no_k_multi                          0.666334               0.017
no_k_multi_use_draw                 0.668634               0.010
k_multi_no_auto_coeff               0.664869               0.011
k_multi_no_auto_coeff_use_draw      0.666442               0.011
k_multi_small_auto_coeff_use_draw   0.664477               0.011
k_multi_multiplicative              0.689797               0.038

                                                             1  \
538_params                          (Boston, 1600.402873295721)
no_k_multi                         (Boston, 1608.3625819770161)
no_k_multi_use_draw                (Boston, 1604.8693089163653)
k_multi_no_auto_coeff              (Boston, 1616.0907500911542)
k_multi_no_auto_coeff_use_draw     (Boston, 1607.5643829924059)
k_multi_small_auto_coeff_use_draw  (Boston, 1601.0646830062885)
k_multi_multiplicative             (Boston, 1656.6599912993656)

                                                                   2  \
538_params                          (NY Rangers, 1557.1728537006936)
no_k_multi                               (Vegas, 1554.5725373751936)
no_k_multi_use_draw                 (Los Angeles, 1560.070774504916)
k_multi_no_auto_coeff                    (Vegas, 1559.5622469426003)
k_multi_no_auto_coeff_use_draw      (Los Angeles, 1559.630863082845)
k_multi_small_auto_coeff_use_draw       (Dallas, 1569.1801789013925)
k_multi_multiplicative             (Los Angeles, 1628.7776039716575)

                                                                   3  \
538_params                               (Vegas, 1556.6388947316857)
no_k_multi                           (NY Rangers, 1553.533247924399)
no_k_multi_use_draw                 (NY Rangers, 1558.8218917594377)
k_multi_no_auto_coeff               (NY Rangers, 1559.3098398839966)
k_multi_no_auto_coeff_use_draw      (NY Rangers, 1559.0834597202997)
k_multi_small_auto_coeff_use_draw  (Los Angeles, 1563.8708258194545)
k_multi_multiplicative              (NY Rangers, 1616.1246575455948)

                                                                   4  \
538_params                         (Los Angeles, 1556.4960245461882)
no_k_multi                            (Colorado, 1553.0269967123975)
no_k_multi_use_draw                      (Vegas, 1556.7951297491593)
k_multi_no_auto_coeff                 (Colorado, 1555.0287306055668)
k_multi_no_auto_coeff_use_draw           (Vegas, 1557.3035611065027)
k_multi_small_auto_coeff_use_draw        (Vegas, 1557.7539099876278)
k_multi_multiplicative                (Edmonton, 1614.2808448386843)

                                                                  5  \
538_params                           (Colorado, 1551.5501005955302)
no_k_multi                         (Los Angeles, 1544.856700412897)
no_k_multi_use_draw                  (Colorado, 1547.8582334504906)
k_multi_no_auto_coeff              (Los Angeles, 1553.622044130705)
k_multi_no_auto_coeff_use_draw         (Dallas, 1557.2622812208258)
k_multi_small_auto_coeff_use_draw  (NY Rangers, 1555.9303720837281)
k_multi_multiplicative                  (Vegas, 1611.0414897840526)

                                                                6  \
538_params                           (Dallas, 1548.5588058968444)
no_k_multi                           (Toronto, 1542.547587411123)
```

```
no_k_multi_use_draw                (Edmonton, 1547.8327469783526)
k_multi_no_auto_coeff                 (Dallas, 1553.072744868073)
k_multi_no_auto_coeff_use_draw     (Edmonton, 1554.2601451080734)
k_multi_small_auto_coeff_use_draw  (Edmonton, 1552.9740681204003)
k_multi_multiplicative             (Winnipeg, 1609.8868246294305)

                                                                7  \
538_params                         (Edmonton, 1545.2393719774434)
no_k_multi                         (Carolina, 1540.6294280917757)
no_k_multi_use_draw                  (Dallas, 1547.5830992968338)
k_multi_no_auto_coeff              (Edmonton, 1551.9744145388693)
k_multi_no_auto_coeff_use_draw     (Colorado, 1548.1610706161414)
k_multi_small_auto_coeff_use_draw  (Colorado, 1541.9531224431455)
k_multi_multiplicative               (Dallas, 1585.6083420985856)

                                                                8  \
538_params                          (Toronto, 1536.6396446872147)
no_k_multi                          (Edmonton, 1539.950874426994)
no_k_multi_use_draw                 (Toronto, 1540.8488001010576)
k_multi_no_auto_coeff               (Toronto, 1546.2475104501286)
k_multi_no_auto_coeff_use_draw      (Toronto, 1542.9307145553937)
k_multi_small_auto_coeff_use_draw   (Toronto, 1541.5184317980422)
k_multi_multiplicative             (Colorado, 1583.2242733679748)

                                                               -4  \
538_params                          (Chicago, 1420.6883124604783)
no_k_multi                           (Chicago, 1428.756329799601)
no_k_multi_use_draw                (San Jose, 1422.2030379206296)
k_multi_no_auto_coeff               (Chicago, 1412.8674499042374)
k_multi_no_auto_coeff_use_draw     (San Jose, 1409.9666261701097)
k_multi_small_auto_coeff_use_draw   (Chicago, 1404.1246463544846)
k_multi_multiplicative              (Chicago, 1372.9955725877526)

                                                               -3  \
538_params                         (Columbus, 1409.6955342651713)
no_k_multi                         (Columbus, 1411.7030157642416)
no_k_multi_use_draw                (Columbus, 1414.6728471195647)
k_multi_no_auto_coeff              (Columbus, 1399.0736633758297)
k_multi_no_auto_coeff_use_draw     (Columbus, 1405.7995418626854)
k_multi_small_auto_coeff_use_draw  (Columbus, 1403.6250618303206)
k_multi_multiplicative               (Seattle, 1346.518639883216)

                                                               -2  \
538_params                         (San Jose, 1407.0529011677697)
no_k_multi                           (Anaheim, 1411.330337901672)
no_k_multi_use_draw                 (Chicago, 1410.1819642175064)
k_multi_no_auto_coeff              (San Jose, 1396.8451096425495)
```

```
k_multi_no_auto_coeff_use_draw       (Chicago, 1404.171567730812)
k_multi_small_auto_coeff_use_draw  (San Jose, 1390.9307099445173)
k_multi_multiplicative             (Columbus, 1333.4113464604384)

                                                              -1
538_params                         (Anaheim, 1406.2583882722054)
no_k_multi                          (San Jose, 1408.95691284118)
no_k_multi_use_draw                (Anaheim, 1398.6759645687955)
k_multi_no_auto_coeff              (Anaheim, 1393.9667371744872)
k_multi_no_auto_coeff_use_draw     (Anaheim, 1391.7382584759303)
k_multi_small_auto_coeff_use_draw  (Anaheim, 1383.3402013173363)
k_multi_multiplicative              (Anaheim, 1311.460800468155)
```

[ ]:

```
predicted_probs_df
```

[ ]:

```
                 date     home_team   away_team  score_diff  \
1742  Mon Dec 11 2023  NY Islanders     Toronto           1
1743  Tue Dec 12 2023    Pittsburgh     Arizona           2
1744  Tue Dec 12 2023        Ottawa    Carolina          -3
1745  Tue Dec 12 2023         Vegas     Calgary           1
1746  Tue Dec 12 2023      Edmonton     Chicago           3
…               …           …         …         …
2024  Sat Jan 20 2024         Vegas  Pittsburgh           1
2025  Sat Jan 20 2024       Buffalo   Tampa Bay          -2
2026  Sat Jan 20 2024     Vancouver     Toronto           2
2027  Sat Jan 20 2024        Ottawa    Winnipeg          -1
2028  Sat Jan 20 2024      St Louis  Washington           3

      home_prob_538_params  home_prob_no_k_multi  \
1742              0.515521              0.476808
1743              0.612933              0.583040
1744              0.516813              0.472941
1745              0.678137              0.657525
1746              0.732001              0.695147
…                    …                   …
2024              0.659692              0.648351
2025              0.537284              0.520602
2026              0.553986              0.508179
2027              0.515257              0.482301
2028              0.563523              0.553479

      home_prob_no_k_multi_use_draw  home_prob_k_multi_no_auto_coeff  \
1742                       0.498510                         0.475904
1743                       0.579436                         0.594386
1744                       0.487689                         0.475061
1745                       0.635349                         0.649821
1746                       0.720674                         0.723511
```

```
…                             …                              …
2024                       0.636626                         0.637445
2025                       0.500249                         0.504113
2026                       0.489094                         0.501283
2027                       0.488420                         0.484927
2028                       0.529172                         0.532669

      home_prob_k_multi_no_auto_coeff_use_draw  \
1742                                  0.498061
1743                                  0.595811
1744                                  0.493392
1745                                  0.630963
1746                                  0.735977
…                                        …
2024                                  0.626369
2025                                  0.496262
2026                                  0.493358
2027                                  0.498497
2028                                  0.521408

      home_prob_k_multi_small_auto_coeff_use_draw  \
1742                                     0.505207
1743                                     0.609128
1744                                     0.512638
1745                                     0.627372
1746                                     0.734588
…                                           …
2024                                     0.603057
2025                                     0.495258
2026                                     0.526498
2027                                     0.508726
2028                                     0.510495

      home_prob_k_multi_multiplicative
1742                          0.425976
1743                          0.516090
1744                          0.518198
1745                          0.774712
1746                          0.840171
…                                …
2024                          0.785477
2025                          0.495491
2026                          0.572753
2027                          0.380103
2028                          0.557855

[287 rows x 11 columns]
```

```
[ ]: indices_to_check = [0,1,2,3,4,5,6,283,284,285,286]  #specific games can inspect
     predicted_probs_df.iloc[indices_to_check,:]
```

```
[ ]:                  date     home_team   away_team  score_diff  \
     1742  Mon Dec 11 2023  NY Islanders     Toronto           1
     1743  Tue Dec 12 2023    Pittsburgh     Arizona           2
     1744  Tue Dec 12 2023        Ottawa    Carolina          -3
     1745  Tue Dec 12 2023         Vegas     Calgary           1
     1746  Tue Dec 12 2023      Edmonton     Chicago           3
     1747  Tue Dec 12 2023      St Louis     Detroit          -2
     1748  Tue Dec 12 2023       Seattle     Florida           4
     2025  Sat Jan 20 2024       Buffalo   Tampa Bay          -2
     2026  Sat Jan 20 2024     Vancouver     Toronto           2
     2027  Sat Jan 20 2024        Ottawa    Winnipeg          -1
     2028  Sat Jan 20 2024      St Louis  Washington           3

           home_prob_538_params  home_prob_no_k_multi  \
     1742              0.515521              0.476808
     1743              0.612933              0.583040
     1744              0.516813              0.472941
     1745              0.678137              0.657525
     1746              0.732001              0.695147
     1747              0.550888              0.552573
     1748              0.485234              0.483006
     2025              0.537284              0.520602
     2026              0.553986              0.508179
     2027              0.515257              0.482301
     2028              0.563523              0.553479

           home_prob_no_k_multi_use_draw  home_prob_k_multi_no_auto_coeff  \
     1742                       0.498510                         0.475904
     1743                       0.579436                         0.594386
     1744                       0.487689                         0.475061
     1745                       0.635349                         0.649821
     1746                       0.720674                         0.723511
     1747                       0.520832                         0.527460
     1748                       0.476643                         0.475150
     2025                       0.500249                         0.504113
     2026                       0.489094                         0.501283
     2027                       0.488420                         0.484927
     2028                       0.529172                         0.532669

           home_prob_k_multi_no_auto_coeff_use_draw  \
     1742                                  0.498061
     1743                                  0.595811
     1744                                  0.493392
     1745                                  0.630963
```

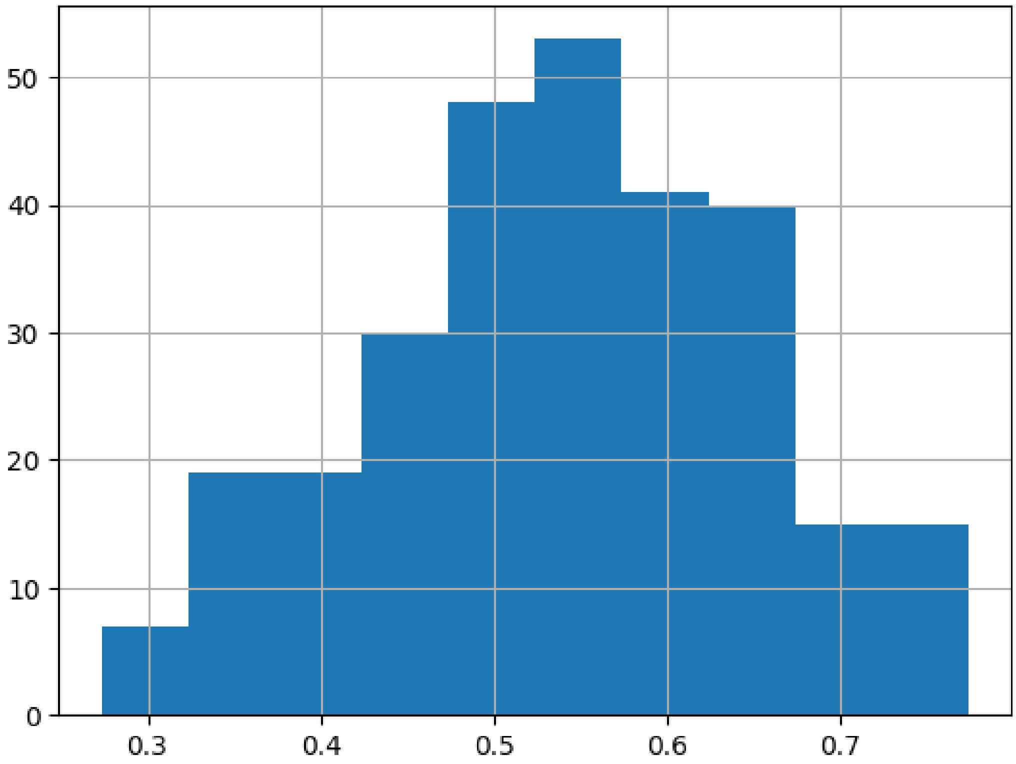
```
1746                                  0.735977
1747                                  0.511767
1748                                  0.482831
2025                                  0.496262
2026                                  0.493358
2027                                  0.498497
2028                                  0.521408

      home_prob_k_multi_small_auto_coeff_use_draw  \
1742                                     0.505207
1743                                     0.609128
1744                                     0.512638
1745                                     0.627372
1746                                     0.734588
1747                                     0.494228
1748                                     0.469512
2025                                     0.495258
2026                                     0.526498
2027                                     0.508726
2028                                     0.510495

      home_prob_k_multi_multiplicative
1742                          0.425976
1743                          0.516090
1744                          0.518198
1745                          0.774712
1746                          0.840171
1747                          0.559005
1748                          0.273902
2025                          0.495491
2026                          0.572753
2027                          0.380103
2028                          0.557855
```

```
[ ]: predicted_probs_df.home_prob_k_multi_small_auto_coeff_use_draw.hist()
```

```
[ ]: <Axes: >
```



[ ]:

```
'''
CHOSEN MODEL : k_multi_small_auto_coeff_use_draw
'''
elo_model = EloClassifier(team_names, 1500, k=12, alpha=0.8, home_adv=28,␣
 ↪mov_exp=1.5, mov_bias=0.3, auto_coeff=0.018, auto_bias=9.2, k_mult='538',␣
 ↪use_draw=True)
elo_model.fit(train_df)
log_loss = elo_model.score(val_df, val_df[['home_win']].values)
print(f"log loss = {log_loss}")
rankings = elo_model.return_sorted_list()
```

```
log loss = 0.6644771975075382
```

0.1 Answers to Questions

[ ]:

```
print(f"1) Expected (independent of context) Best = {rankings[0]}, \n Expected␣
 ↪Worst = {rankings[-1]}")
probs = elo_model.predict_proba( pd.DataFrame([[0,'Nashville', 'Florida']],␣
 ↪columns=['numeric_season', 'home_team', 'away_team']) , update_ratings=False)
print(f"2) Florida Win_Prob (as away team) against NSH = {np.round(probs[0,1],␣
 ↪3)}")
boston_ranking = rankings[0][1]
```

```
boston_win_prob_against_avg_team = elo_model.win_prob(boston_ranking, np.
 ↪mean(list(elo_model.elo_scores_dict.values())))    # average ranking 1500 by␣
 ↪construction
print(f"3) Expect Boston to win {np.round(boston_win_prob_against_avg_team*100,␣
 ↪2)}% of their remaining matches")
```

```
1) Expected (indepedent of context) Best = ('Boston', 1601.0646830062885),
 Expected Worst = ('Anaheim', 1383.3402013173363)
2) Florida Win_Prob (as away team) against NSH = 0.526
3) Expect Boston to win 64.16% of their remaining matches
```

```
[ ]: #######################
```