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

from sklearn.linear_model import RidgeClassifier
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
```

In [6]:
```
df = pd.read_csv("nba_games_data.csv", index_col=0)
```

In [7]:
```
df = df.sort_values("date")
df = df.reset_index(drop= True)
```

In [8]:
```
del df["mp.1"]
del df["mp_opp.1"]
del df["index_opp"]
```

In [9]:
```
def add_target(team):
    team["target"] = team["won"].shift(-1)
    return team

df = df.groupby("team", group_keys=False).apply(add_target)
```

```
/var/folders/df/st36jx9n2xnfdgt9qr2ymcnh0000gn/T/ipykernel_10710/1127067056.
py:2: PerformanceWarning: DataFrame is highly fragmented.  This is usually t
he result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get
a de-fragmented frame, use `newframe = frame.copy()`
  team["target"] = team["won"].shift(-1)
/var/folders/df/st36jx9n2xnfdgt9qr2ymcnh0000gn/T/ipykernel_10710/1127067056.
py:2: PerformanceWarning: DataFrame is highly fragmented.  This is usually t
he result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead. To get
a de-fragmented frame, use `newframe = frame.copy()`
  team["target"] = team["won"].shift(-1)
/var/folders/df/st36jx9n2xnfdgt9qr2ymcnh0000gn/T/ipykernel_10710/1127067056.
py:5: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping co
lumns. This behavior is deprecated, and in a future version of pandas the gr
ouping columns will be excluded from the operation. Either pass `include_gro
ups=False` to exclude the groupings or explicitly select the grouping column
s after groupby to silence this warning.
  df = df.groupby("team", group_keys=False).apply(add_target)
```

In [10]:
```
df["target"][pd.isnull(df["target"])] = 2
df["target"] = df["target"].astype(int, errors="ignore")
```

```
/var/folders/df/st36jx9n2xnfdgt9qr2ymcnh0000gn/T/ipykernel_10710/2400510588.
py:1: FutureWarning: ChainedAssignmentError: behaviour will change in pandas
3.0!
You are setting values through chained assignment. Currently this works in c
ertain cases, but when using Copy-on-Write (which will become the default be
haviour in pandas 3.0) this will never work to update the original DataFrame
or Series, because the intermediate object on which we are setting values wi
ll behave as a copy.
A typical example is when you are setting values in a column of a DataFrame,
like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment
in a single step and ensure this keeps updating the original `df`.

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

  df["target"][pd.isnull(df["target"])] = 2
/var/folders/df/st36jx9n2xnfdgt9qr2ymcnh0000gn/T/ipykernel_10710/2400510588.
py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["target"][pd.isnull(df["target"])] = 2
```

In [11]:
```
nulls = pd.isnull(df).sum()
```

In [12]:
```
nulls = nulls[nulls > 0]
```

In [13]:
```
nulls
```

Out[13]:
```
+/-             9748
mp_max          9748
mp_max.1        9748
+/-_max           15
+/-_opp         9748
mp_max_opp      9748
mp_max_opp.1    9748
+/-_max_opp       15
dtype: int64
```

In [14]:
```
valid_columns = df.columns[~df.columns.isin(nulls.index)]
```

In [15]:
```
valid_columns
```

Out[15]:
```
Index(['mp', 'fg', 'fga', 'fg%', '3p', '3pa', '3p%', 'ft', 'fta', 'ft%',
       ...
       'usg%_max_opp', 'ortg_max_opp', 'drtg_max_opp', 'team_opp', 'total_o
pp',
       'home_opp', 'season', 'date', 'won', 'target'],
      dtype='object', length=140)
```

In [16]:
```
df = df[valid_columns].copy()
```

In [17]:
```
#Feature Selection
```

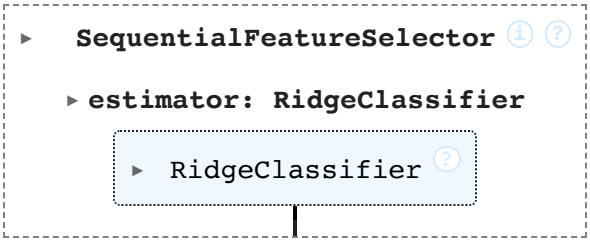
In [18]:
```
rr = RidgeClassifier(alpha=1)

split = TimeSeriesSplit(n_splits=3)

sfs = SequentialFeatureSelector(rr,
                                n_features_to_select=25,
                                direction="forward",
                                cv=split,
                                n_jobs=1
                               )
```

In [19]:
```
removed_columns = ["season", "date", "won", "target", "team", "team_opp"]
selected_columns = df.columns[~df.columns.isin(removed_columns)]
```

In [20]:
```
scaler = MinMaxScaler()
df[selected_columns] = scaler.fit_transform(df[selected_columns])
```

In [21]:
```
sfs.fit(df[selected_columns], df["target"])
```

Out[21]:



In [22]:
```
predictors = list(selected_columns[sfs.get_support()])
```

In [23]:
```
predictors
```

Out[23]:
```
['fga',
 'ft',
 'pts',
 'efg%',
 'usg%',
 'fga_max',
 'tov_max',
 'efg%_max',
 'stl%_max',
 'blk%_max',
 'tov%_max',
 'ortg_max',
 'total',
 '3p%_opp',
 'pf_opp',
 'pts_opp',
 'usg%_opp',
 'fg_max_opp',
 'ft%_max_opp',
 'blk_max_opp',
 'pf_max_opp',
 'pts_max_opp',
 'blk%_max_opp',
 'usg%_max_opp',
 'total_opp']
```

In [27]:
```
def backtest(data, model, predictors, start=2, step=1):
    all_predictions = []

    seasons = sorted(data["season"].unique())

    for i in range(start, len(seasons), step):
        season = seasons[i]
        train = data[data["season"] < season]
        test = data[data["season"] == season]

        model.fit(train[predictors], train["target"])

        preds = model.predict(test[predictors])
        preds = pd.Series(preds, index=test.index)
        combined = pd.concat([test["target"], preds], axis=1)
        combined.columns = ["actual", "prediction"]

        all_predictions.append(combined)
    return pd.concat(all_predictions)
```

In [30]:
```
predictions = backtest(df, rr, predictors)
```

In [31]:
```
accuracy_score(predictions["actual"], predictions["prediction"])
```

Out[31]: 0.5520242149073024

In [49]:
```
df_rolling = df[list(selected_columns) + ["won", "team", "season"]]
```

In [50]:
```
df_rolling
```

Out[50]:

	mp	fg	fga	fg%	3p	3pa	3p%	
0	0.333333	0.500000	0.672414	0.366029	0.629630	0.660377	0.568369	0.365
1	0.000000	0.477273	0.310345	0.598086	0.333333	0.396226	0.457990	0.365
2	0.000000	0.363636	0.379310	0.397129	0.407407	0.433962	0.522241	0.317
3	0.333333	0.477273	0.689655	0.332536	0.444444	0.566038	0.449753	0.731
4	0.000000	0.431818	0.465517	0.418660	0.148148	0.264151	0.285008	0.292
...	...	...	...	...	...	...	...	
9743	0.000000	0.295455	0.500000	0.241627	0.333333	0.471698	0.390445	0.317
9744	0.000000	0.318182	0.258621	0.430622	0.222222	0.283019	0.400329	0.365
9745	0.000000	0.409091	0.275862	0.538278	0.444444	0.339623	0.696870	0.341
9746	0.000000	0.272727	0.568966	0.179426	0.259259	0.471698	0.296540	0.292
9747	0.000000	0.386364	0.362069	0.437799	0.111111	0.339623	0.168040	0.268

9748 rows × 137 columns

In [ ]:
```
def find_team_averages(team):
    rolling = team.rolling(10).mean()
    return rolling

df_rolling = df_rolling.groupby(["team", "season"], group_keys=False).apply(
```

In [ ]:
```
def shift_col(team, col_name):
    next_col = team[col_name].shift(-1)
    return next_col

def add_col(df, col_name):
    return df.groupby("team", group_keys=False).apply(lambda x: shift_col(x,

df["home_next"] = add_col(df, "home")
df["team_opp_next"] = add_col(df, "team_opp")
df["date_next"] = add_col(df, "date")
```

In [ ]:
```
removed_columns = list(full.columns[full.dtypes == "object"]) + removed_colu
```

In [ ]:
```
selected_columns = full.columns[~full.columns.isin(removed_columns)]
sfs.fit(full[selected_columns], full["target"])
```

In [ ]:
```
predictors = list(selected_columns[sfs.get_support()])
```

In [ ]:
```
predictions = backtest(full, rr, predictors)
```

In [ ]:
```
accuracy_score(predictions["actual"], predictions["prediction"])
```