Capstone Project: NCAA Basketball Tournament Predictions

DSTC 691: Default Machine Learning Project

Name: Garrett Fanning

Introduction

The National Collegiate Basketball Association (NCAA) has over 350 colleges all competing to earn a spot in the NCAA tournament, with the ultimate goal of winning the tournament. Only 68 colleges are able to play well enough to earn a spot in the tournament. The lowest 8 teams play against each other to whittle down the field to 64 teams, from which a typical bracket style tournament can be created.

For the most part, picking the winner of each game comes down to guessing or using your general basketball knowledge where it's impossible to consistently predict games correctly. The NCAA tournament at first glance may appear to have a lot of randomness, which has led many people to leverage the vast amount of college basketball data out there to create algorithms that can have more predictability than the simple guessing or just picking the consensus favorite for each game.

I will use regular season data to help predict how that translates to success in the NCAA tournament. All of the features I will be using are averages or on a per game basis because all teams don't play an equal amount of games in the regular season. Many of these features will not be used if they do not appear significant or are too similar to other features. I will predict winners of matchups in the NCAA tournament through predicting a final score differential, which can be interpreted as a positive differential means one team wins or negative would mean the other team.

I was able to acquire the necessary data through 2 sources: the sportsreference website and through a csv uploaded by someone performing a similar project on the data.world website. The first source (sportsreference website) actually has a library in sklearn that is able to pull data directly through the site. From there I was able to pull 10 years of regular season data with 40 columns and 3478 rows of data. The second source(data.world) requires you to make an account on the site to actually download the csv. I was able to get tournament data dating back to 1985 with 2205 rows and 10 columns of data. Merging these datasets to eventually get to the relvant feature set was difficult where many of the team names that were supposed to match were actually different between the datasets. After cleaning and merging the data, I then did some exploratory analysis to narrow down the features to 29.

Those 29 features were scaled and then trained using different machine learning models with final score difference being the response variable. The regression models I used are Gradient Boosting, Random Forest, Decision Tree, K-Nearest Neigbors, Voting Ensemble of those 4 previous models, and ANN with tensorflow. Using MSE as a performance metric to determining the best performing model. The benchmark models I created to compare against were a model predicting by the better seed and the other predicting by the better record.

Flask was then used to deploy the best performing model. Using the Flask application, the feature data for a a tournament game matchup is fed to the application and a prediction on the final score difference will be provided using the best performing model.

Data Description

Source #1: https://www.sports-reference.com/cbb/

This dataset has 10 years of data with give or take 350 teams of season data per year.Some teams were added in the time period of the data resulting in an uneven amount of teams with data per year. For each team looking at a single year, I collected 40 variables of data that were either statistical game averages or rankings/ratings they were given or earned in that same year. The csv file that I downloaded from the API that directly pulls data from the sportsreference website was 789KB. Definition of each column can be found here: https://www.sports-reference.com/cbb/about/glossary.html

Source #2: https://data.world/michaelaroy/ncaa-tournament-results/workspace/file?filename=Big_Dance_CSV.csv

This dataset has tournament games dating back to 1985. For the range of data used in the models, only games from 2010-2019 are relevant. The only feature from this file are the seeds and the response variable of final score difference is from this dataset. The full dataset downloaded from the website has 2205 rows and 10 columns of data. The csv file size is 101KB.

Libraries

General Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from sportsreference.ncaab.teams import Teams

from statsmodels.api import OLS


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.metrics import mean_squared_error

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

import pickle
import os
import flask
from flask import Flask, redirect, url_for,request, render_template
import math
```

Table of Contents install

In [ ]:
```
#pip install --user jupyter_contrib_nbextensions
#jupyter contrib nbextension install --user
#jupyter nbextension enable toc2/main
```

API to download data from sports reference website

Doesn't need to be run because data from here was stored in a csv, so that I wouldn't have to repeat this process each time I opened the notebook. The csv file will be provided in the Google Drive.

In [63]:
```
# #pip install pandas sklearn sportsreference  --> in terminal

# #Source: https://www.sports-reference.com/cbb/

# #!!!The code below will take minimum 5 minutes to run

# #to see what data/features are available
# #teams2019.dataframes.columns

# #creating an empty dataset to be filled
# #all columns except for those with totals as all teams don't play the same amount of games

# dataset = pd.DataFrame(columns = ['year','name','abbreviation', 'assist_percentage', 'block_percentage',
#                                   'effective_field_goal_percentage',
#                                   'field_goal_percentage', 'free_throw_attempt_rate', 'free_throw_percentage',
#                                   'free_throws_per_field_goal_attempt',
#                                   'offensive_rating', 'offensive_rebound_percentage',
#                                   'opp_assist_percentage',
#                                   'opp_block_percentage',
#                                   'opp_effective_field_goal_percentage','opp_field_goal_percentage',
#                                   'opp_free_throw_attempt_rate',
#                                   'opp_free_throw_percentage',
#                                   'opp_free_throws_per_field_goal_attempt', 'opp_offensive_rating',
#                                   'opp_offensive_rebound_percentage',
#                                   'opp_steal_percentage',
#                                   'opp_three_point_attempt_rate','opp_three_point_field_goal_percentage',
#                                   'opp_two_point_field_goal_percentage','opp_total_rebound_percentage',
#                                   'opp_true_shooting_percentage', 'opp_turnover_percentage',
#                                   'pace', 'simple_rating_system', 'steal_percentage',
#                                   'strength_of_schedule', 'three_point_attempt_rate',
#                                   'three_point_field_goal_percentage',
#                                   'two_point_field_goal_percentage', 'two_point_field_goals',
#                                   'total_rebound_percentage',
#                                   'true_shooting_percentage', 'turnover_percentage','win_percentage'])
# for yearVal in range(2010,2020,1):

#     team = Teams(year = str(yearVal))

#     temp = {'year':[str(yearVal)]*sum(team.dataframes['name'].value_counts()), 'name':team.dataframes.name,'abbreviation':team.dataframes.abbreviation,
#                             'assist_percentage':team.dataframes.assist_percentage,'block_percentage':team.dataframes.block_percentage,
#                             'effective_field_goal_percentage' : team.dataframes.effective_field_goal_percentage,
#                             'field_goal_percentage':team.dataframes.field_goal_percentage, 'free_throw_attempt_rate':team.dataframes.free_throw_attempt_rate,
#                             'free_throw_percentage':team.dataframes.free_throw_percentage,
#                             'free_throws_per_field_goal_attempt':team.dataframes.free_throws_per_field_goal_attempt, 'offensive_rating': team.dataframes.offensive_
#                             'offensive_rebound_percentage':team.dataframes.offensive_rebound_percentage,'opp_assist_percentage':team.dataframes.opp_assist_percenta
#                             'opp_block_percentage': team.dataframes.opp_block_percentage,'opp_effective_field_goal_percentage':team.dataframes.opp_effective_field_
#                             'opp_field_goal_percentage':team.dataframes.opp_field_goal_percentage, 'opp_free_throw_attempt_rate':team.dataframes.opp_free_throw_att
#                             'opp_free_throw_percentage': team.dataframes.opp_free_throw_percentage,
#                             'opp_free_throws_per_field_goal_attempt': team.dataframes.opp_free_throws_per_field_goal_attempt, 'opp_offensive_rating':team.datafram
#                             'opp_offensive_rebound_percentage': team.dataframes.opp_offensive_rebound_percentage,
#                             'opp_steal_percentage': team.dataframes.opp_steal_percentage,
#                             'opp_three_point_attempt_rate': team.dataframes.opp_three_point_attempt_rate,
#                             'opp_three_point_field_goal_percentage' : team.dataframes.opp_three_point_field_goal_percentage,
#                             'opp_two_point_field_goal_percentage': team.dataframes.opp_two_point_field_goal_percentage,
#                             'opp_total_rebound_percentage' : team.dataframes.opp_total_rebound_percentage,
#                             'opp_true_shooting_percentage':  team.dataframes.opp_true_shooting_percentage, 'opp_turnover_percentage':team.dataframes.opp_turnover_p
#                             'pace': team.dataframes.pace, 'simple_rating_system': team.dataframes.simple_rating_system, 'steal_percentage': team.dataframes.steal_p
#                             'strength_of_schedule': team.dataframes.strength_of_schedule, 'three_point_attempt_rate':team.dataframes.three_point_attempt_rate,
#                             'three_point_field_goal_percentage': team.dataframes.three_point_field_goal_percentage,
#                             'two_point_field_goal_percentage': team.dataframes.two_point_field_goal_percentage, 'two_point_field_goals':team.dataframes.two_point_j
```

```
#                           'total_rebound_percentage': team.dataframes.total_rebound_percentage,'true_shooting_percentage':team.dataframes.true_shooting_percentag
#                           'turnover_percentage': team.dataframes.turnover_percentage, 'win_percentage':team.dataframes.win_percentage}
#     df_temp = pd.DataFrame(data=temp)

#     dataset = dataset.append(df_temp,ignore_index=True)

# dataset.to_csv('season_results.csv')
# dataset
```

Out[63]:

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_f
0	2010	Air Force	AIR-FORCE	61.6	6.2	0.504	0.443	0.367	0.635	
1	2010	Akron	AKRON	53.9	8.5	0.491	0.433	0.363	0.657	
2	2010	Alabama A&M	ALABAMA-AM	48.1	12.7	0.416	0.382	0.474	0.635	
3	2010	UAB	ALABAMA-BIRMINGHAM	51.1	7.3	0.471	0.422	0.457	0.694	
4	2010	Alabama State	ALABAMA-STATE	60.0	11.1	0.462	0.404	0.448	0.641	
...	...	...	...	...	...	...	...	...	...	
3473	2019	Wright State	WRIGHT-STATE	54.7	6.1	0.506	0.436	0.341	0.737	
3474	2019	Wyoming	WYOMING	48.0	7.9	0.492	0.417	0.399	0.723	
3475	2019	Xavier	XAVIER	56.3	10.6	0.528	0.466	0.326	0.679	
3476	2019	Yale	YALE	56.3	11.2	0.556	0.493	0.307	0.738	
3477	2019	Youngstown State	YOUNGSTOWN-STATE	51.1	9.9	0.501	0.427	0.246	0.701	

3478 rows × 40 columns

Import Data

Import season results from csv file created through API

This dataset has 10 years of data with give or take 350 teams of season data per year.Some teams were added in the time period of the data resulting in an uneven amount of teams with data per year. For each team looking at a single year, I collected 40 variables of data that were either statistical game averages or rankings/ratings they were given or earned in that same year. The csv file that I downloaded from the API that directly pulls data from the sportsreference website was 101KB.

In [2]:

```
#Import data from file named below

fileName = "season_results.csv"

season_results = pd.read_csv(fileName)

#Drop index column, year should be the first column with real data
```

```
season_results = season_results.iloc[: , 1:]

season_results
```

Out[2]:

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_f
0	2010	Air Force	AIR-FORCE	61.6	6.2	0.504	0.443	0.367	0.635	
1	2010	Akron	AKRON	53.9	8.5	0.491	0.433	0.363	0.657	
2	2010	Alabama A&M	ALABAMA-AM	48.1	12.7	0.416	0.382	0.474	0.635	
3	2010	UAB	ALABAMA-BIRMINGHAM	51.1	7.3	0.471	0.422	0.457	0.694	
4	2010	Alabama State	ALABAMA-STATE	60.0	11.1	0.462	0.404	0.448	0.641	
...	...	...	...	...	...	...	...	...	...	
3473	2019	Wright State	WRIGHT-STATE	54.7	6.1	0.506	0.436	0.341	0.737	
3474	2019	Wyoming	WYOMING	48.0	7.9	0.492	0.417	0.399	0.723	
3475	2019	Xavier	XAVIER	56.3	10.6	0.528	0.466	0.326	0.679	
3476	2019	Yale	YALE	56.3	11.2	0.556	0.493	0.307	0.738	
3477	2019	Youngstown State	YOUNGSTOWN-STATE	51.1	9.9	0.501	0.427	0.246	0.701	

3478 rows × 40 columns

In [3]:

```
#Get statistics on each column
season_results.describe()
```

Out[3]:

	year	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_goal_attempt	off
count	3478.000000	3478.000000	3478.000000	3478.000000	3478.000000	3478.000000	3478.000000	3478.000000	
mean	2014.537378	52.964232	9.377343	0.498732	0.438739	0.364334	0.697412	0.254014	
std	2.864129	5.291287	2.653611	0.031027	0.025613	0.051574	0.037643	0.037891	
min	2010.000000	34.700000	2.900000	0.397000	0.347000	0.210000	0.541000	0.141000	
25%	2012.000000	49.300000	7.400000	0.478000	0.422000	0.329000	0.672000	0.228000	
50%	2015.000000	52.900000	9.200000	0.499000	0.439000	0.363000	0.698000	0.253000	
75%	2017.000000	56.475000	11.100000	0.520000	0.456000	0.398000	0.723000	0.279000	
max	2019.000000	74.000000	20.400000	0.605000	0.526000	0.593000	0.818000	0.415000	

8 rows × 38 columns

In [4]:

```
#Some percentages were listed as 50.0 instead of .500
#To be consistent with all features, any feature with percentage in the name will be less than 1
season_results['assist_percentage'] = season_results['assist_percentage']/100
season_results['block_percentage'] = season_results['block_percentage']/100
season_results['offensive_rebound_percentage'] = season_results['offensive_rebound_percentage']/100
season_results['opp_assist_percentage'] = season_results['opp_assist_percentage']/100
season_results['opp_block_percentage'] = season_results['opp_block_percentage']/100
season_results['opp_offensive_rebound_percentage'] = season_results['opp_offensive_rebound_percentage']/100
season_results['opp_steal_percentage'] = season_results['opp_steal_percentage']/100
season_results['opp_total_rebound_percentage'] = season_results['opp_total_rebound_percentage']/100
season_results['opp_turnover_percentage'] = season_results['opp_turnover_percentage']/100
season_results['steal_percentage'] = season_results['steal_percentage']/100
season_results['opp_turnover_percentage'] = season_results['opp_turnover_percentage']/100
season_results['total_rebound_percentage'] = season_results['total_rebound_percentage']/100
season_results['turnover_percentage'] = season_results['turnover_percentage']/100
```

In [5]:

```
season_results.describe()
```

Out[5]:

	year	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_goal_attempt	off
count	3478.000000	3478.000000	3478.000000	3478.000000	3478.000000	3478.000000	3478.000000	3478.000000	
mean	2014.537378	0.529642	0.093773	0.498732	0.438739	0.364334	0.697412	0.254014	
std	2.864129	0.052913	0.026536	0.031027	0.025613	0.051574	0.037643	0.037891	
min	2010.000000	0.347000	0.029000	0.397000	0.347000	0.210000	0.541000	0.141000	
25%	2012.000000	0.493000	0.074000	0.478000	0.422000	0.329000	0.672000	0.228000	
50%	2015.000000	0.529000	0.092000	0.499000	0.439000	0.363000	0.698000	0.253000	
75%	2017.000000	0.564750	0.111000	0.520000	0.456000	0.398000	0.723000	0.279000	
max	2019.000000	0.740000	0.204000	0.605000	0.526000	0.593000	0.818000	0.415000	

8 rows × 38 columns

In [6]:

```
#See if there are any missing cells of data
season_results.isnull().sum()
```

Out[6]:

```
year                                0
name                                0
abbreviation                        0
assist_percentage                   0
block_percentage                    0
effective_field_goal_percentage     0
field_goal_percentage               0
free_throw_attempt_rate             0
free_throw_percentage               0
free_throws_per_field_goal_attempt  0
offensive_rating                    0
offensive_rebound_percentage        0
opp_assist_percentage               0
opp_block_percentage                0
```

```
opp_effective_field_goal_percentage         0
opp_field_goal_percentage                   0
opp_free_throw_attempt_rate                 0
opp_free_throw_percentage                   0
opp_free_throws_per_field_goal_attempt      0
opp_offensive_rating                     3478
opp_offensive_rebound_percentage            0
opp_steal_percentage                        0
opp_three_point_attempt_rate                0
opp_three_point_field_goal_percentage       0
opp_two_point_field_goal_percentage         0
opp_total_rebound_percentage                0
opp_true_shooting_percentage                0
opp_turnover_percentage                     0
pace                                        0
simple_rating_system                        0
steal_percentage                            0
strength_of_schedule                        0
three_point_attempt_rate                    0
three_point_field_goal_percentage           0
two_point_field_goal_percentage             0
two_point_field_goals                       0
total_rebound_percentage                    0
true_shooting_percentage                    0
turnover_percentage                         0
win_percentage                              0
dtype: int64
```

In [7]:

```
# Completely empty column and so it is impossible to impute values to fill it up
season_results=season_results.drop(columns=['opp_offensive_rating'])
```

Import Tournament Results Data from csv file found online

In [8]:

```
#Import data from file named below
#Source: https://data.world/michaelaroy/ncaa-tournament-results/workspace/file?filename=Big_Dance_CSV.csv

fileName = "Big_Dance_CSV.csv"

tourney_results = pd.read_csv(fileName)
tourney_results
```

Out[8]:

	Year	Round	Region Number	Region Name	Seed	Score	Team	Team.1	Score.1	Seed.1
0	1985	1	1	West	1	83	St Johns	Southern	59	16
1	1985	1	1	West	2	81	VCU	Marshall	65	15
2	1985	1	1	West	3	65	NC State	Nevada	56	14
3	1985	1	1	West	4	85	UNLV	San Diego St	80	13
4	1985	1	1	West	5	58	Washington	Kentucky	65	12
...	...	...	...	...	...	...	...	...	...	...
2200	2019	4	3	East	1	80	Virginia	Purdue	75	3
2201	2019	4	4	Midwest	5	77	Auburn	Kentucky	71	2
2202	2019	5	1	Final Four	2	51	Michigan St	Texas Tech	61	3

	Year	Round	Region Number	Region Name	Seed	Score	Team	Team.1	Score.1	Seed.1
2203	2019	5	2	Final Four	1	63	Virginia	Auburn	62	5
2204	2019	6	1	Championship	3	77	Texas Tech	Virginia	85	1

2205 rows × 10 columns

In [9]:

```
tourney_results.describe()
```

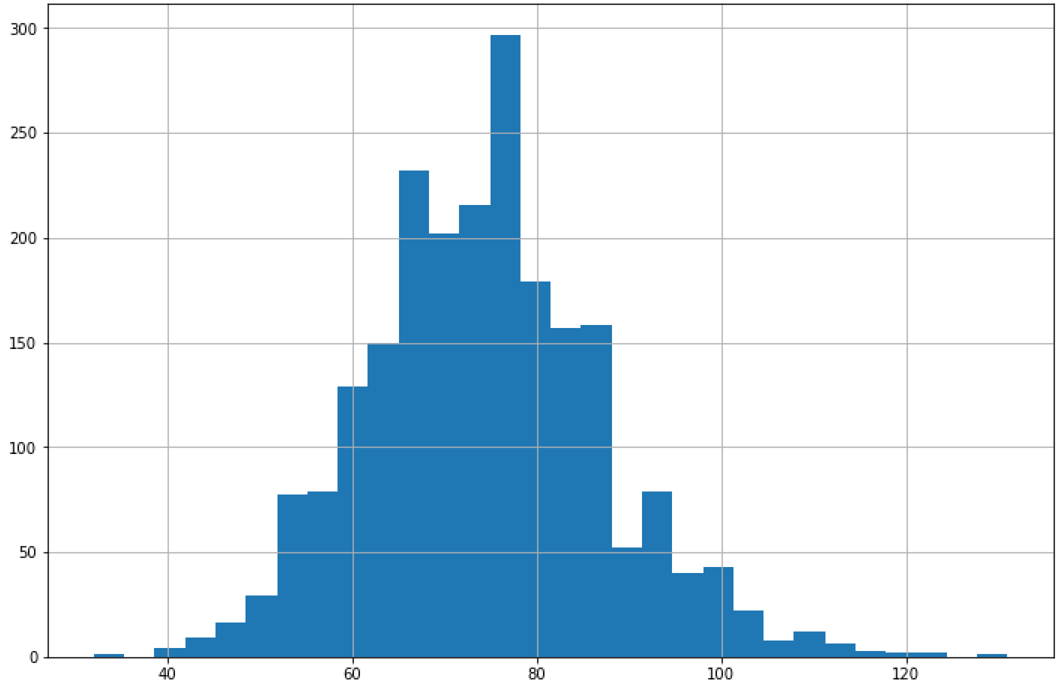
Out[9]:

	Year	Round	Region Number	Seed	Score	Score.1	Seed.1
count	2205.000000	2205.000000	2205.000000	2205.000000	2205.000000	2205.000000	2205.000000
mean	2002.000000	1.904762	2.444444	3.887528	74.304308	68.051701	9.474830
std	10.101796	1.191698	1.123993	2.900662	12.753399	12.295443	4.138256
min	1985.000000	1.000000	1.000000	1.000000	32.000000	29.000000	1.000000
25%	1993.000000	1.000000	1.000000	2.000000	65.000000	59.000000	6.000000
50%	2002.000000	1.000000	2.000000	3.000000	74.000000	67.000000	10.000000
75%	2011.000000	2.000000	3.000000	6.000000	82.000000	76.000000	13.000000
max	2019.000000	6.000000	4.000000	16.000000	131.000000	149.000000	16.000000

In [10]:

```
#See distribution of score column that will be used in the calculation of the response variable
tourney_results['Score'].hist(bins=30, figsize=(12, 8))
```

Out[10]: <AxesSubplot:>



In [11]:
```
#Looking into unusually high score found
tourney_results.iloc[tourney_results['Score'].idxmax()]
```

Out[11]:
```
Year                        1990
Round                          4
Region Number                  2
Region Name                 West
Seed                           1
Score                        131
Team                        UNLV
Team.1          Loyola Marymount
Score.1                      101
Seed.1                        11
Name: 372, dtype: object
```

Confirmed on the sportsreference website at : https://www.sports-reference.com/cbb/boxscores/1990-03-25-loyola-marymount.html

Nevada-Las Vegas vs. Loyola Marymount Box Score, March 25, 1990

NCAA Tournament Scores — Mar 25, 1990

MINN (6)	91	Final	UNLV (1)	131	Fin
GT (4)	93		LOYMR (11)	101	

UNLV
131

Loyola Marymount
101

« Prev Game Next Game » « Prev Game

March 25, 1990
Oracle Arena, Oakland, California
West - Regional Final
Logos via Sports Logos.net / About logos

In [12]:

```
#Confirm there is no missing values
tourney_results.isnull().sum()
```

Out[12]:

```
Year             0
Round            0
Region Number    0
Region Name      0
Seed             0
Score            0
Team             0
Team.1           0
Score.1          0
Seed.1           0
dtype: int64
```

In [13]:

```
#Select columns needed from dataframe created using the csv file
tourney_results = pd.DataFrame(tourney_results,columns =['Year','Seed','Score','Team','Team.1','Score.1','Seed.1'])

#Only need tournament games from 2010-2019
tourney_results_features = tourney_results[(tourney_results['Year'] > 2009) & (tourney_results['Year'] < 2020)]
tourney_results_features
```

Out[13]:

	Year	Seed	Score	Team	Team.1	Score.1	Seed.1
1575	2010	1	90	Kansas	Lehigh	74	16
1576	2010	2	68	Ohio St	Santa Barbara	51	15
1577	2010	3	83	Georgetown	Ohio	97	14

	Year	Seed	Score	Team	Team.1	Score.1	Seed.1
1578	2010	4	89	Maryland	Houston	77	13
1579	2010	5	70	Michigan St	New Mexico St	67	12
...	...	...	...	...	...	...	...
2200	2019	1	80	Virginia	Purdue	75	3
2201	2019	5	77	Auburn	Kentucky	71	2
2202	2019	2	51	Michigan St	Texas Tech	61	3
2203	2019	1	63	Virginia	Auburn	62	5
2204	2019	3	77	Texas Tech	Virginia	85	1

630 rows × 7 columns

Merge Season Results & Tourney Results DataFrames

The goal here is to first get the 2 data sources(Tournament Results & Season Results) into the right state to be merged together. The process will be to merge the season results of the home team in the tournament games into 1 DataFrame. The other DataFrame would be the merge of the away team. Then simply taking the Home Team Dataframe and subracting from it the Away Team DataFrame would give the required Season Statistic Differentials that could be used as the features with the final game score differential in each tournament game being the response. Using the data like this would accomplish my initial goal of trying to find/establish a relationship between how teams performed in the regular season would lead to success in the tournament.

Rename team names in tournament results DataFrame to match with correct team names in season result DataFrame

In [14]:

```
# Using exported csvs from both data sources, exported csvs stored in Google Drive
# Compared both datasets to compare and change team names to match
# Manually change ' to \' for 5 team names

tourney_results_features = tourney_results_features.replace({'Alabama St':'Alabama State','Albany':'Albany (NY)','Alcorn St':'Alcorn State','Arizona St':'Arizona Stat
                                  'Arkansas Little Rock':'Little Rock','Arkansas Pine Bluff':'Arkansas-Pine Bluff','Ball St':'Ball State','Boise St':'Boise State',
                                  'BYU':'Brigham Young','Cal Irvine':'UC Irvine','Cal St Bakersfield':'Cal State Bakersfield', 'Cal St Fullerton':'Cal State Fullerton'
                                  'Central Connecticut St':'Central Connecticut State','Cleveland St':'Cleveland State',
                                     'Colorado St':'Colorado State',
                                     'Coppin St':'Coppin State',
                                     'Delaware St':'Delaware State',
                                     'Detroit':'Detroit Mercy',
                                     'East Tennessee St':'East Tennessee State',
                                     'Florida St':'Florida State',
                                     'Fresno St':'Fresno State',
                                     'Gardner Webb':'Gardner-Webb',
                                     'Georgia St':'Georgia State',
                                     'Idaho St':'Idaho State',
                                     'Illinois Chicago':'Illinois-Chicago',
                                     'Illinois St':'Illinois State',
                                     'Indiana St':'Indiana State',
                                     'Iowa St':'Iowa State',
                                     'Jackson St':'Jackson State',
                                     'Jacksonville St':'Jacksonville State',
                                     'Kansas St':'Kansas State',
                                     'Kent St':'Kent State',
```

```
'Long Beach St':'Long Beach State',
'Long Island Brooklyn':'Long Island University',
'Louisiana Lafayette':'Lafayette',
'Louisiana Monroe':'Louisiana-Monroe',
'Loyola Chicago':'Loyola (IL)',
'Loyola Illinois':'Loyola (IL)',
'Loyola Maryland':'Loyola (MD)',
'LSU':'Louisiana State',
'McNeese St':'McNeese State',
'Miami':'Miami (FL)',
'Miami Ohio':'Miami (OH)',
'Michigan St':'Michigan State',
'Middle Tennessee St':'Middle Tennessee',
'Mississippi St':'Mississippi State',
'Mississippi Valley St':'Mississippi Valley State',
'Montana St':'Montana State',
'Morehead St':'Morehead State',
'Morgan St':'Morgan State',
'Mount St Marys':'Mount St. Mary\'s',
'Murray St':'Murray State',
'New Mexico St':'New Mexico State',
'Nicholls St':'Nicholls State',
'Norfolk St':'Norfolk State',
'North Dakota St':'North Dakota State',
'North Texas St':'North Texas',
'Northwestern St':'Northwestern State',
'Ohio St':'Ohio State',
'Oklahoma St':'Oklahoma State',
'Ole Miss':'Mississippi',
'Oregon St':'Oregon State',
'Penn St':'Penn State',
'Portland St':'Portland State',
'Sam Houston St':'Sam Houston State',
'San Diego St':'San Diego State',
'San Jose St':'San Jose State',
'Santa Barbara':'UC Santa Barbara',
'SMU':'Southern Methodist',
'South Carolina St':'South Carolina State',
'South Dakota St':'South Dakota State',
'Southeast Missouri St':'Southeast Missouri State',
'Southwest Missouri St':'Missouri State',
'Southwest Texas St':'Texas State',
'St Bonaventure':'St. Bonaventure',
'St Francis':'Saint Francis (PA)',
'St Johns':'St. John\'s (NY)',
'St Josephs':'Saint Joseph\'s',
'St Louis':'Saint Louis',
'St Marys':'Saint Mary\'s (CA)',
'St Peters':'Saint Peter\'s',
'Stephen F Austin':'Stephen F. Austin',
'Tennessee St':'Tennessee State',
'Texas A&M Corpus Christi':'Texas A&M-Corpus Christi',
'Texas Arlington':'UT Arlington',
'Texas San Antonio':'UTSA',
'UMBC':'Maryland-Baltimore County',
'UNLV':'Nevada-Las Vegas',
'USC':'Southern California',
'Utah St':'Utah State',
'VCU':'Virginia Commonwealth',
'Washington St':'Washington State',
```

```
                                        'Weber St':'Weber State',
                                        'Wichita St':'Wichita State',
                                        'Wisconsin Green Bay':'Green Bay',
                                        'Wisconsin Milwaukee':'Milwaukee',
                                        'Wright St':'Wright State'})
```

In [ ]:

```
# #DO NOT RUN
# #This was used to find initial differences in datasets for team names
# #Exported team name lists to csv and then manually scanned to see what changes needed to be made

# merged_dataset_away['Away_Team'].value_counts()

# counts_SeasonResults = dataset.value_counts(['name']) #counts_SeasonResults.to_csv('counts_SeasonResults.csv')

# counts_TourneyResults = tourney_results_features.value_counts(['Home_Team','Away_Team']) counts_TourneyResults.to_csv('counts_TourneyResults_v2.csv')

# counts_merged_away = merged_dataset_away.value_counts(['Away_Team']) counts_merged_away.to_csv('counts_merged_away.csv')

# counts_merged_home = merged_dataset_home.value_counts(['Home_Team']) counts_merged_home.to_csv('counts_merged_home.csv')
```

Prepare DataFrames to be merged

In [15]:

```
#Rename features
tourney_results_features = tourney_results_features.rename(columns={"Seed":"Home_Seed","Score":"Home_Score","Team":"Home_Team","Team.1":"Away_Team",
                                        "Score.1":"Away_Score","Seed.1":"Away_Seed"})
```

In [16]:

```
season_results['name'].value_counts()
```

Out[16]:

```
Campbell                10
South Florida           10
Coastal Carolina        10
North Texas             10
Montana State           10
                        ..
Incarnate Word           6
Massachusetts-Lowell     6
Centenary (LA)           2
California Baptist       1
North Alabama            1
Name: name, Length: 354, dtype: int64
```

In [17]:

```
tourney_results_features['Home_Team'].value_counts()
```

Out[17]:

```
Kentucky               31
Kansas                 30
Duke                   27
North Carolina         25
Michigan State         21
                       ..
Illinois                1
Loyola (IL)             1
Lehigh                  1
Wofford                 1
Southern California     1
Name: Home_Team, Length: 109, dtype: int64
```

In [18]:
```
#Looking into dtypes for both datasets to make sure matching columns also have matching dtypes
tourney_results.dtypes
```

Out[18]:
```
Year        int64
Seed        int64
Score       int64
Team       object
Team.1     object
Score.1     int64
Seed.1      int64
dtype: object
```

In [19]:
```
season_results.dtypes
```

Out[19]:
```
year                                    int64
name                                   object
abbreviation                           object
assist_percentage                     float64
block_percentage                      float64
effective_field_goal_percentage       float64
field_goal_percentage                 float64
free_throw_attempt_rate               float64
free_throw_percentage                 float64
free_throws_per_field_goal_attempt    float64
offensive_rating                      float64
offensive_rebound_percentage          float64
opp_assist_percentage                 float64
opp_block_percentage                  float64
opp_effective_field_goal_percentage   float64
opp_field_goal_percentage             float64
opp_free_throw_attempt_rate           float64
opp_free_throw_percentage             float64
opp_free_throws_per_field_goal_attempt    float64
opp_offensive_rebound_percentage      float64
opp_steal_percentage                  float64
opp_three_point_attempt_rate          float64
opp_three_point_field_goal_percentage float64
opp_two_point_field_goal_percentage   float64
opp_total_rebound_percentage          float64
opp_true_shooting_percentage          float64
opp_turnover_percentage               float64
pace                                  float64
simple_rating_system                  float64
steal_percentage                      float64
strength_of_schedule                  float64
three_point_attempt_rate              float64
three_point_field_goal_percentage     float64
two_point_field_goal_percentage       float64
two_point_field_goals                   int64
total_rebound_percentage              float64
true_shooting_percentage              float64
turnover_percentage                   float64
win_percentage                        float64
dtype: object
```

In [20]:
```
#Initially had issues with merging because season_results was a different data type between the 2 sources
season_results['year'] = pd.to_numeric(season_results['year'])
season_results.dtypes
```

Out[20]:
```
year                                    int64
name                                   object
abbreviation                           object
```

```
assist_percentage                       float64
block_percentage                        float64
effective_field_goal_percentage         float64
field_goal_percentage                   float64
free_throw_attempt_rate                 float64
free_throw_percentage                   float64
free_throws_per_field_goal_attempt      float64
offensive_rating                        float64
offensive_rebound_percentage            float64
opp_assist_percentage                   float64
opp_block_percentage                    float64
opp_effective_field_goal_percentage     float64
opp_field_goal_percentage               float64
opp_free_throw_attempt_rate             float64
opp_free_throw_percentage               float64
opp_free_throws_per_field_goal_attempt  float64
opp_offensive_rebound_percentage        float64
opp_steal_percentage                    float64
opp_three_point_attempt_rate            float64
opp_three_point_field_goal_percentage   float64
opp_two_point_field_goal_percentage     float64
opp_total_rebound_percentage            float64
opp_true_shooting_percentage            float64
opp_turnover_percentage                 float64
pace                                    float64
simple_rating_system                    float64
steal_percentage                        float64
strength_of_schedule                    float64
three_point_attempt_rate                float64
three_point_field_goal_percentage       float64
two_point_field_goal_percentage         float64
two_point_field_goals                     int64
total_rebound_percentage                float64
true_shooting_percentage                float64
turnover_percentage                     float64
win_percentage                          float64
dtype: object
```

Merge home team tournament games to the season results in that year

In [21]:

```
#Merge home team tournament games to the season results in that year
merged_dataset_home = pd.merge(season_results,tourney_results_features,how='inner',left_on=['year','name'],right_on=['Year','Home_Team'],validate="1:m")
merged_dataset_home
```

Out[21]:

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_g
0	2010	Baylor	BAYLOR	0.512	0.174	0.549	0.487	0.358	0.725	
1	2010	Baylor	BAYLOR	0.512	0.174	0.549	0.487	0.358	0.725	
2	2010	Brigham Young	BRIGHAM-YOUNG	0.553	0.091	0.552	0.483	0.388	0.790	
3	2010	Butler	BUTLER	0.549	0.065	0.510	0.442	0.469	0.738	
4	2010	Butler	BUTLER	0.549	0.065	0.510	0.442	0.469	0.738	
...	...	...	...	...	...	...	...	...	...	
625	2019	Virginia	VIRGINIA	0.559	0.130	0.552	0.474	0.291	0.744	

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_g
626	2019	Virginia	VIRGINIA	0.559	0.130	0.552	0.474	0.291	0.744	
627	2019	Virginia	VIRGINIA	0.559	0.130	0.552	0.474	0.291	0.744	
628	2019	Wisconsin	WISCONSIN	0.492	0.110	0.511	0.449	0.273	0.648	
629	2019	Wofford	WOFFORD	0.506	0.091	0.580	0.490	0.271	0.704	

630 rows × 46 columns

In [22]:
```
#Confirm merge dataset rows equal to amount of games in 2010-2019 tourney results dataframe
len(tourney_results_features)
```

Out[22]:
```
630
```

In [23]:
```
merged_dataset_home.isnull().sum()
```

Out[23]:
```
year                                 0
name                                 0
abbreviation                         0
assist_percentage                    0
block_percentage                     0
effective_field_goal_percentage      0
field_goal_percentage                0
free_throw_attempt_rate              0
free_throw_percentage                0
free_throws_per_field_goal_attempt   0
offensive_rating                     0
offensive_rebound_percentage         0
opp_assist_percentage                0
opp_block_percentage                 0
opp_effective_field_goal_percentage  0
opp_field_goal_percentage            0
opp_free_throw_attempt_rate          0
opp_free_throw_percentage            0
opp_free_throws_per_field_goal_attempt 0
opp_offensive_rebound_percentage     0
opp_steal_percentage                 0
opp_three_point_attempt_rate         0
opp_three_point_field_goal_percentage 0
opp_two_point_field_goal_percentage  0
opp_total_rebound_percentage         0
opp_true_shooting_percentage         0
opp_turnover_percentage              0
pace                                 0
simple_rating_system                 0
steal_percentage                     0
strength_of_schedule                 0
three_point_attempt_rate             0
three_point_field_goal_percentage    0
two_point_field_goal_percentage      0
two_point_field_goals                0
total_rebound_percentage             0
true_shooting_percentage             0
turnover_percentage                  0
win_percentage                       0
```

```
Year                        0
Home_Seed                   0
Home_Score                  0
Home_Team                   0
Away_Team                   0
Away_Score                  0
Away_Seed                   0
dtype: int64
```

Merge away team tournament games to the season results in that year

In [24]:

```
merged_dataset_away = pd.merge(season_results,tourney_results_features,how='inner',left_on=['year','name'],right_on=['Year','Away_Team'],validate="1:m")
merged_dataset_away
```

Out[24]:

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_fielc
0	2010	Arkansas-Pine Bluff	ARKANSAS-PINE-BLUFF	0.581	0.102	0.447	0.408	0.484	0.668	
1	2010	Baylor	BAYLOR	0.512	0.174	0.549	0.487	0.358	0.725	
2	2010	Baylor	BAYLOR	0.512	0.174	0.549	0.487	0.358	0.725	
3	2010	Brigham Young	BRIGHAM-YOUNG	0.553	0.091	0.552	0.483	0.388	0.790	
4	2010	Butler	BUTLER	0.549	0.065	0.510	0.442	0.469	0.738	
...	...	...	...	...	...	...	...	...	...	
625	2019	Virginia	VIRGINIA	0.559	0.130	0.552	0.474	0.291	0.744	
626	2019	Washington	WASHINGTON	0.475	0.163	0.521	0.451	0.347	0.695	
627	2019	Washington	WASHINGTON	0.475	0.163	0.521	0.451	0.347	0.695	
628	2019	Wofford	WOFFORD	0.506	0.091	0.580	0.490	0.271	0.704	
629	2019	Yale	YALE	0.563	0.112	0.556	0.493	0.307	0.738	

630 rows × 46 columns

Sort merge away and merge home to make sure same games are in correct order for both DataFrames before finding differences

In [25]:

```
merged_dataset_away = merged_dataset_away.sort_values(by=['year', 'Home_Team','Away_Team'])
merged_dataset_away = merged_dataset_away.reset_index(drop=True)
merged_dataset_away
```

Out[25]:

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_g

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_g
0	2010	Old Dominion	OLD-DOMINION	0.601	0.086	0.488	0.447	0.303	0.648	
1	2010	Sam Houston State	SAM-HOUSTON-STATE	0.740	0.070	0.536	0.463	0.370	0.703	
2	2010	Florida	FLORIDA	0.538	0.072	0.494	0.444	0.334	0.703	
3	2010	Duke	DUKE	0.529	0.098	0.505	0.442	0.379	0.759	
4	2010	Kansas State	KANSAS-STATE	0.551	0.132	0.508	0.450	0.503	0.668	
...	...	...	...	...	...	...	...	...	...	
625	2019	Central Florida	CENTRAL-FLORIDA	0.536	0.126	0.529	0.465	0.453	0.649	
626	2019	Liberty	LIBERTY	0.552	0.081	0.569	0.487	0.265	0.775	
627	2019	Saint Louis	SAINT-LOUIS	0.538	0.119	0.467	0.417	0.394	0.598	
628	2019	Oregon	OREGON	0.523	0.146	0.520	0.451	0.294	0.721	
629	2019	Seton Hall	SETON-HALL	0.517	0.112	0.499	0.439	0.345	0.706	

630 rows × 46 columns

In [26]:

```python
merged_dataset_home = merged_dataset_home.sort_values(by=['year', 'Home_Team','Away_Team'])
merged_dataset_home = merged_dataset_home.reset_index(drop=True)
merged_dataset_home
```

Out[26]:

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_p
0	2010	Baylor	BAYLOR	0.512	0.174	0.549	0.487	0.358	0.725	
1	2010	Baylor	BAYLOR	0.512	0.174	0.549	0.487	0.358	0.725	
2	2010	Brigham Young	BRIGHAM-YOUNG	0.553	0.091	0.552	0.483	0.388	0.790	
3	2010	Butler	BUTLER	0.549	0.065	0.510	0.442	0.469	0.738	
4	2010	Butler	BUTLER	0.549	0.065	0.510	0.442	0.469	0.738	
...	...	...	...	...	...	...	...	...	...	
625	2019	Virginia Commonwealth	VIRGINIA-COMMONWEALTH	0.549	0.122	0.501	0.438	0.357	0.701	
626	2019	Virginia Tech	VIRGINIA-TECH	0.595	0.084	0.556	0.470	0.319	0.761	
627	2019	Virginia Tech	VIRGINIA-TECH	0.595	0.084	0.556	0.470	0.319	0.761	

	year	name	abbreviation	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_p
628	2019	Wisconsin	WISCONSIN	0.492	0.110	0.511	0.449	0.273	0.648	
629	2019	Wofford	WOFFORD	0.506	0.091	0.580	0.490	0.271	0.704	

630 rows × 46 columns

In [27]:

```
# Splitting out this data as it's not needed when finding difference/analysis between features of the merged DataFrames
game_info = merged_dataset_home[['Year','Home_Team','Away_Team']]
game_info
```

Out[27]:

	Year	Home_Team	Away_Team
0	2010	Baylor	Old Dominion
1	2010	Baylor	Sam Houston State
2	2010	Brigham Young	Florida
3	2010	Butler	Duke
4	2010	Butler	Kansas State
...	...	...	...
625	2019	Virginia Commonwealth	Central Florida
626	2019	Virginia Tech	Liberty
627	2019	Virginia Tech	Saint Louis
628	2019	Wisconsin	Oregon
629	2019	Wofford	Seton Hall

630 rows × 3 columns

In [28]:

```
#Difference in points that will be used as response
response = merged_dataset_home['Home_Score']-merged_dataset_home['Away_Score']

response
```

Out[28]:

```
0       8
1       9
2       7
3      -2
4       7
       ..
625   -15
626     9
627    14
628   -18
629    16
Length: 630, dtype: int64
```

In [29]:

```
#Difference in seeds that will be a feature added into the feature set used
```

```
seed_differential = merged_dataset_home['Home_Seed']-merged_dataset_home['Away_Seed']

seed_differential
```

Out[29]:

```
0      -8
1     -11
2      -3
3       4
4       3
       ..
625    -1
626    -8
627    -9
628    -7
629    -3
Length: 630, dtype: int64
```

In [30]:

```
merged_dataset_home=merged_dataset_home.drop(columns=['year','name','abbreviation','Year','Home_Seed','Home_Score','Home_Team','Away_Team','Away_Score','Away_Seed'])
merged_dataset_home
```

Out[30]:

	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_goal_attempt	offensive_rating	o
0	0.512	0.174	0.549	0.487	0.358	0.725	0.260	113.8	
1	0.512	0.174	0.549	0.487	0.358	0.725	0.260	113.8	
2	0.553	0.091	0.552	0.483	0.388	0.790	0.307	116.1	
3	0.549	0.065	0.510	0.442	0.469	0.738	0.346	106.8	
4	0.549	0.065	0.510	0.442	0.469	0.738	0.346	106.8	
...	...	...	...	...	...	...	...	...	
625	0.549	0.122	0.501	0.438	0.357	0.701	0.251	102.1	
626	0.595	0.084	0.556	0.470	0.319	0.761	0.243	113.3	
627	0.595	0.084	0.556	0.470	0.319	0.761	0.243	113.3	
628	0.492	0.110	0.511	0.449	0.273	0.648	0.177	103.9	
629	0.506	0.091	0.580	0.490	0.271	0.704	0.190	119.2	

630 rows × 36 columns

In [31]:

```
merged_dataset_away=merged_dataset_away.drop(columns=['year','name','abbreviation','Year','Home_Seed','Home_Score','Home_Team','Away_Team','Away_Score','Away_Seed'])
merged_dataset_away
```

Out[31]:

	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_goal_attempt	offensive_rating	o
0	0.601	0.086	0.488	0.447	0.303	0.648	0.197	106.2	
1	0.740	0.070	0.536	0.463	0.370	0.703	0.260	111.5	
2	0.538	0.072	0.494	0.444	0.334	0.703	0.235	107.4	
3	0.529	0.098	0.505	0.442	0.379	0.759	0.287	115.7	

	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_goal_attempt	offensive_rating	o
4	0.551	0.132	0.508	0.450	0.503	0.668	0.336	110.5	
...	...	...	...	...	...	...	...	...	
625	0.536	0.126	0.529	0.465	0.453	0.649	0.294	107.0	
626	0.552	0.081	0.569	0.487	0.265	0.775	0.205	112.8	
627	0.538	0.119	0.467	0.417	0.394	0.598	0.236	99.4	
628	0.523	0.146	0.520	0.451	0.294	0.721	0.212	106.2	
629	0.517	0.112	0.499	0.439	0.345	0.706	0.243	104.1	

630 rows × 36 columns

In [32]:

```
#Features created from the difference of away and home teams season results
features_differentials = merged_dataset_home-merged_dataset_away.values
features_differentials
```

Out[32]:

	assist_percentage	block_percentage	effective_field_goal_percentage	field_goal_percentage	free_throw_attempt_rate	free_throw_percentage	free_throws_per_field_goal_attempt	offensive_rating	o
0	-0.089	0.088	0.061	0.040	0.055	0.077	0.063	7.6	
1	-0.228	0.104	0.013	0.024	-0.012	0.022	0.000	2.3	
2	0.015	0.019	0.058	0.039	0.054	0.087	0.072	8.7	
3	0.020	-0.033	0.005	0.000	0.090	-0.021	0.059	-8.9	
4	-0.002	-0.067	0.002	-0.008	-0.034	0.070	0.010	-3.7	
...	...	...	...	...	...	...	...	...	
625	0.013	-0.004	-0.028	-0.027	-0.096	0.052	-0.043	-4.9	
626	0.043	0.003	-0.013	-0.017	0.054	-0.014	0.038	0.5	
627	0.057	-0.035	0.089	0.053	-0.075	0.163	0.007	13.9	
628	-0.031	-0.036	-0.009	-0.002	-0.021	-0.073	-0.035	-2.3	
629	-0.011	-0.021	0.081	0.051	-0.074	-0.002	-0.053	15.1	

630 rows × 36 columns

In [33]:

```
features_differentials['seed_difference'] = seed_differential
features_differentials['seed_difference']
```

Out[33]:

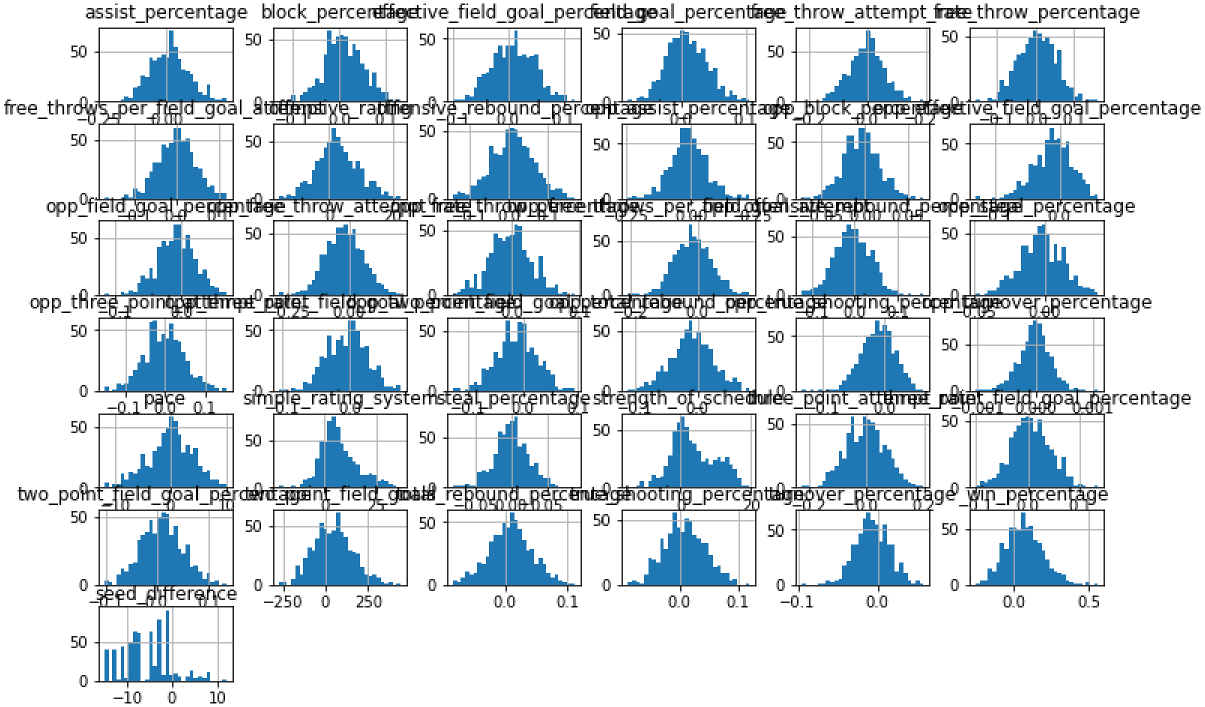
```
0      -8
1     -11
2      -3
```

```
3       4
4       3
       ..
625    -1
626    -8
627    -9
628    -7
629    -3
Name: seed_difference, Length: 630, dtype: int64
```

Do exploratory analysis on features in after merged dataset

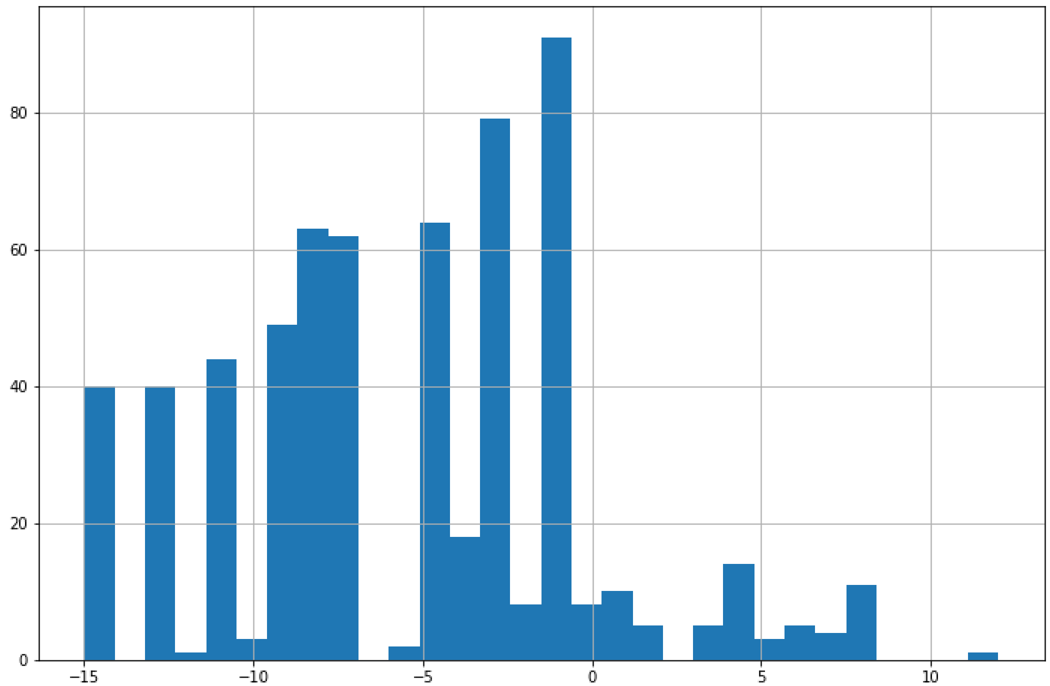
In [34]:
```
features_differentials.hist(bins=30, figsize=(12, 8))
#looking at the histrograms, the features all roughly follow a normal distribution
```

Out[34]:
```
array([[<AxesSubplot:title={'center':'assist_percentage'}>,
        <AxesSubplot:title={'center':'block_percentage'}>,
        <AxesSubplot:title={'center':'effective_field_goal_percentage'}>,
        <AxesSubplot:title={'center':'field_goal_percentage'}>,
        <AxesSubplot:title={'center':'free_throw_attempt_rate'}>,
        <AxesSubplot:title={'center':'free_throw_percentage'}>],
       [<AxesSubplot:title={'center':'free_throws_per_field_goal_attempt'}>,
        <AxesSubplot:title={'center':'offensive_rating'}>,
        <AxesSubplot:title={'center':'offensive_rebound_percentage'}>,
        <AxesSubplot:title={'center':'opp_assist_percentage'}>,
        <AxesSubplot:title={'center':'opp_block_percentage'}>,
        <AxesSubplot:title={'center':'opp_effective_field_goal_percentage'}>],
       [<AxesSubplot:title={'center':'opp_field_goal_percentage'}>,
        <AxesSubplot:title={'center':'opp_free_throw_attempt_rate'}>,
        <AxesSubplot:title={'center':'opp_free_throw_percentage'}>,
        <AxesSubplot:title={'center':'opp_free_throws_per_field_goal_attempt'}>,
        <AxesSubplot:title={'center':'opp_offensive_rebound_percentage'}>,
        <AxesSubplot:title={'center':'opp_steal_percentage'}>],
       [<AxesSubplot:title={'center':'opp_three_point_attempt_rate'}>,
        <AxesSubplot:title={'center':'opp_three_point_field_goal_percentage'}>,
        <AxesSubplot:title={'center':'opp_two_point_field_goal_percentage'}>,
        <AxesSubplot:title={'center':'opp_total_rebound_percentage'}>,
        <AxesSubplot:title={'center':'opp_true_shooting_percentage'}>,
        <AxesSubplot:title={'center':'opp_turnover_percentage'}>],
       [<AxesSubplot:title={'center':'pace'}>,
        <AxesSubplot:title={'center':'simple_rating_system'}>,
        <AxesSubplot:title={'center':'steal_percentage'}>,
        <AxesSubplot:title={'center':'strength_of_schedule'}>,
        <AxesSubplot:title={'center':'three_point_attempt_rate'}>,
        <AxesSubplot:title={'center':'three_point_field_goal_percentage'}>],
       [<AxesSubplot:title={'center':'two_point_field_goal_percentage'}>,
        <AxesSubplot:title={'center':'two_point_field_goals'}>,
        <AxesSubplot:title={'center':'total_rebound_percentage'}>,
        <AxesSubplot:title={'center':'true_shooting_percentage'}>,
        <AxesSubplot:title={'center':'turnover_percentage'}>,
        <AxesSubplot:title={'center':'win_percentage'}>],
       [<AxesSubplot:title={'center':'seed_difference'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]],
      dtype=object)
```



In [35]:
```python
#Only feature that is not normally distributed, which the skewness makes sense because the home team(usally the lower seed/better team) is more likely to advance and
features_differentials['seed_difference'].hist(bins=30, figsize=(12, 8))
```

Out[35]: <AxesSubplot:>



In [36]:

```
features_differentials['seed_difference'].describe()
```

Out[36]:

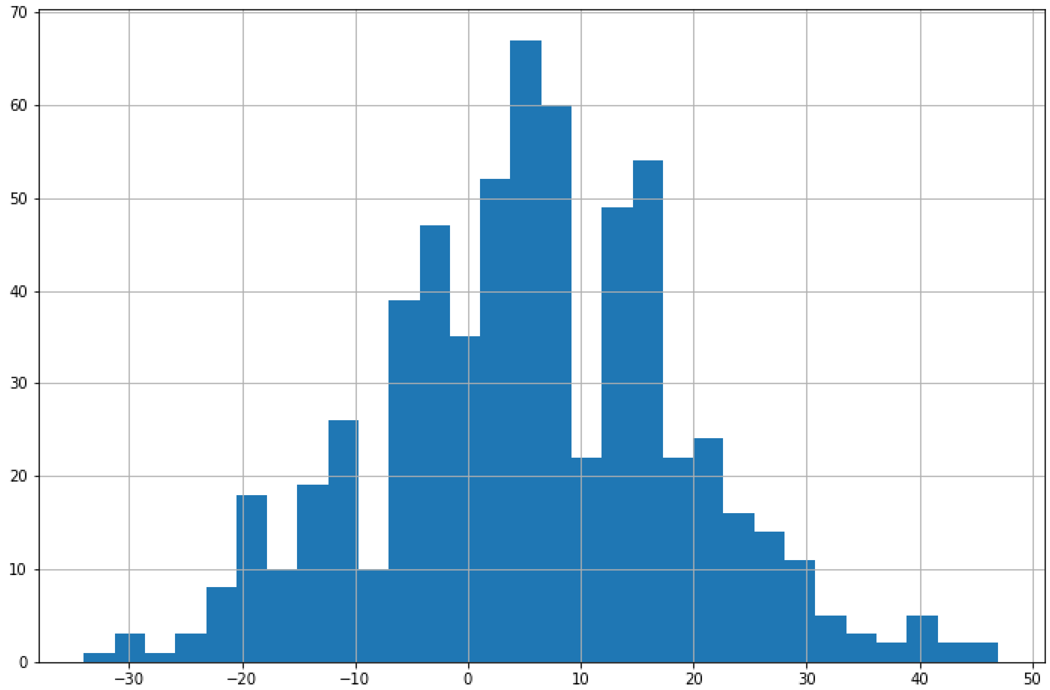
```
count    630.000000
mean      -5.569841
std        5.261739
min      -15.000000
25%       -9.000000
50%       -5.000000
75%       -2.000000
max       12.000000
Name: seed_difference, dtype: float64
```

In [37]:

```
#Confirming reponse(score difference) makes sense and show mainly close games
response.hist(bins=30, figsize=(12, 8))
```

Out[37]:

```
<AxesSubplot:>
```

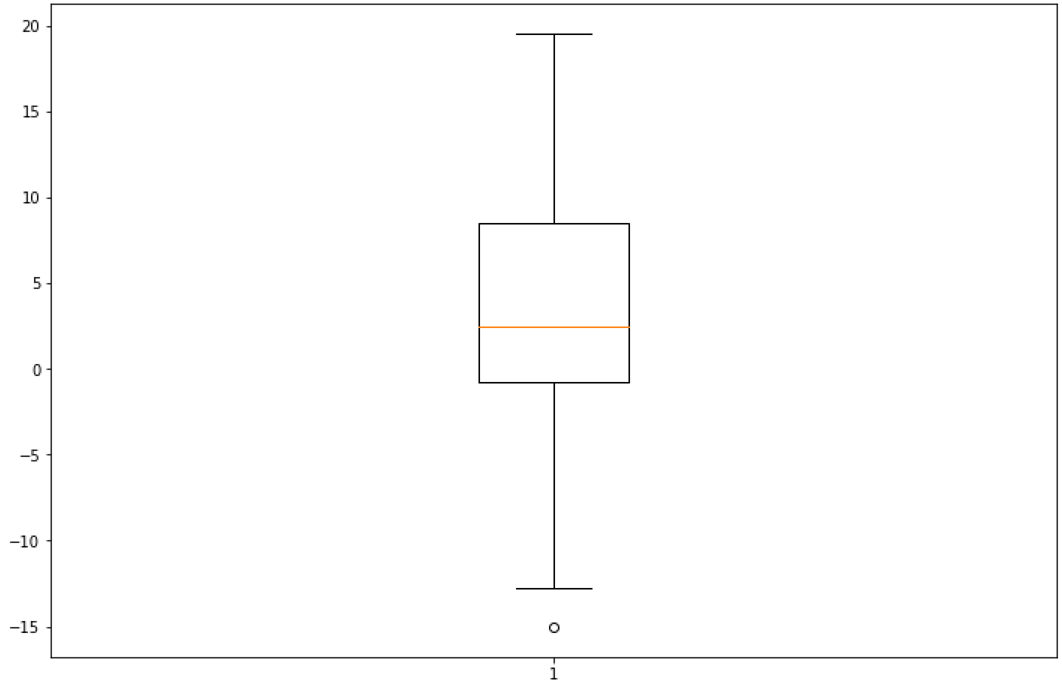


In [38]:

```
fig = plt.figure(figsize =(12, 8))

# Creating plot
plt.boxplot(features_differentials['strength_of_schedule'])

#Scaling might make sense because features reflecting % difference on diffete scale than features such as strength of schedule with much higher difference
```

Out[38]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1b5f08d5be0>,
  <matplotlib.lines.Line2D at 0x1b5ee9aa9a0>],
 'caps': [<matplotlib.lines.Line2D at 0x1b5ef575bb0>,
  <matplotlib.lines.Line2D at 0x1b5f0a06610>],
 'boxes': [<matplotlib.lines.Line2D at 0x1b5f0838a30>],
 'medians': [<matplotlib.lines.Line2D at 0x1b5f0a0c670>],
 'fliers': [<matplotlib.lines.Line2D at 0x1b5ee999c40>],
 'means': []}
```



High level and brief look into significance and correlation of possible features to be used

In [39]:

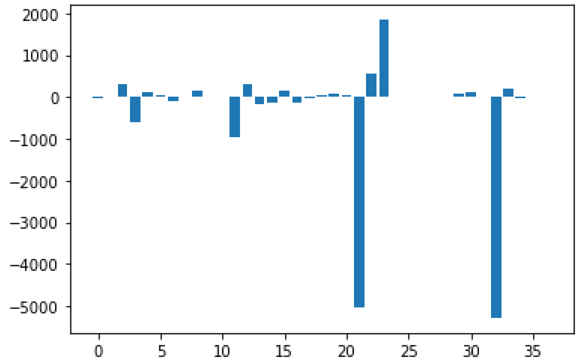
```
#Using Linear Regression for a quick peek at coefficients if current feature set is used in the regression

initial_model = LinearRegression()

initial_model.fit(features_differentials, response)

print('coefficients:', initial_model.coef_)
```

```
coefficients: [-1.33607618e+01  3.22644002e-01  2.93523402e+02 -6.16955601e+02
  1.08753896e+02  5.05948140e+01 -1.10691725e+02 -9.01442385e-01
  1.58029176e+02 -6.02907253e+00  9.46155653e+00 -9.46357446e+02
  3.18007344e+02 -1.69773439e+02 -1.49978144e+02  1.52682540e+02
 -1.41696441e+02 -2.91429095e+01  6.56152316e+01  7.35410643e+01
  4.95037445e+01 -5.02749370e+03  5.59350057e+02  1.84273052e+03
 -1.10859128e+00  3.14594756e-01  2.77665443e+01  7.04248532e-01
  6.99537143e+00  9.83687502e+01  1.15923815e+02  9.65019465e-02
 -5.28740809e+03  2.07483512e+02 -1.15353377e+01  1.21708795e+01
  4.61385685e-01]
```

In [40]:

```
#Using coefficients to get a rough view of feature importance
LinReg_importance = initial_model.coef_
plt.bar([x for x in range(len(LinReg_importance))],LinReg_importance)
plt.show
```

Out[40]: <function matplotlib.pyplot.show(close=None, block=None)>



In [41]:
```
OLS(response,features_differentials).fit().summary()
```

Out[41]:

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	0.606
Model:	OLS	**Adj. R-squared (uncentered):**	0.581
Method:	Least Squares	**F-statistic:**	24.62
Date:	Sat, 23 Apr 2022	**Prob (F-statistic):**	4.82e-96
Time:	21:30:36	**Log-Likelihood:**	-2285.9
No. Observations:	630	**AIC:**	4646.
Df Residuals:	593	**BIC:**	4810.
Df Model:	37		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
assist_percentage	-13.5106	6.447	-2.096	0.037	-26.172	-0.849
block_percentage	0.1451	15.114	0.010	0.992	-29.538	29.828
effective_field_goal_percentage	284.6147	525.781	0.541	0.588	-748.004	1317.233
field_goal_percentage	-601.3151	499.300	-1.204	0.229	-1581.927	379.297
free_throw_attempt_rate	111.6183	112.941	0.988	0.323	-110.194	333.431
free_throw_percentage	52.4844	58.206	0.902	0.368	-61.830	166.799
free_throws_per_field_goal_attempt	-115.8065	176.829	-0.655	0.513	-463.094	231.481
offensive_rating	-0.8429	0.916	-0.920	0.358	-2.641	0.956
offensive_rebound_percentage	150.5522	93.097	1.617	0.106	-32.288	333.392
opp_assist_percentage	-5.9503	8.278	-0.719	0.473	-22.208	10.307
opp_block_percentage	11.5401	22.581	0.511	0.610	-32.808	55.888
opp_effective_field_goal_percentage	-984.9155	518.665	-1.899	0.058	-2003.559	33.728
opp_field_goal_percentage	367.2397	552.470	0.665	0.506	-717.795	1452.275

opp_free_throw_attempt_rate	-169.7085	168.485	-1.007	0.314	-500.609	161.192
opp_free_throw_percentage	-149.8185	77.593	-1.931	0.054	-302.209	2.572
opp_free_throws_per_field_goal_attempt	153.4094	257.187	0.596	0.551	-351.699	658.518
opp_offensive_rebound_percentage	-137.0293	77.838	-1.760	0.079	-289.900	15.842
opp_steal_percentage	-28.4637	44.555	-0.639	0.523	-115.969	59.042
opp_three_point_attempt_rate	73.3286	87.430	0.839	0.402	-98.381	245.038
opp_three_point_field_goal_percentage	78.5765	86.510	0.908	0.364	-91.326	248.479
opp_two_point_field_goal_percentage	46.2667	133.233	0.347	0.729	-215.400	307.933
opp_total_rebound_percentage	-4775.4105	6844.410	-0.698	0.486	-1.82e+04	8666.822
opp_true_shooting_percentage	556.0650	376.553	1.477	0.140	-183.476	1295.606
opp_turnover_percentage	1817.9955	5063.455	0.359	0.720	-8126.492	1.18e+04
pace	-1.1033	0.149	-7.423	0.000	-1.395	-0.811
simple_rating_system	0.3318	0.417	0.795	0.427	-0.488	1.151
steal_percentage	29.1303	35.284	0.826	0.409	-40.166	98.426
strength_of_schedule	0.7214	0.464	1.556	0.120	-0.189	1.632
three_point_attempt_rate	8.4207	87.418	0.096	0.923	-163.267	180.108
three_point_field_goal_percentage	96.1881	76.146	1.263	0.207	-53.361	245.737
two_point_field_goal_percentage	112.4929	115.022	0.978	0.328	-113.407	338.393
two_point_field_goals	0.0953	0.009	11.204	0.000	0.079	0.112
total_rebound_percentage	-5024.8247	6844.598	-0.734	0.463	-1.85e+04	8417.778
true_shooting_percentage	196.8160	385.671	0.510	0.610	-560.632	954.264
turnover_percentage	-5.0203	134.354	-0.037	0.970	-268.888	258.847
win_percentage	12.4589	6.548	1.903	0.058	-0.400	25.318
seed_difference	0.5545	0.153	3.632	0.000	0.255	0.854

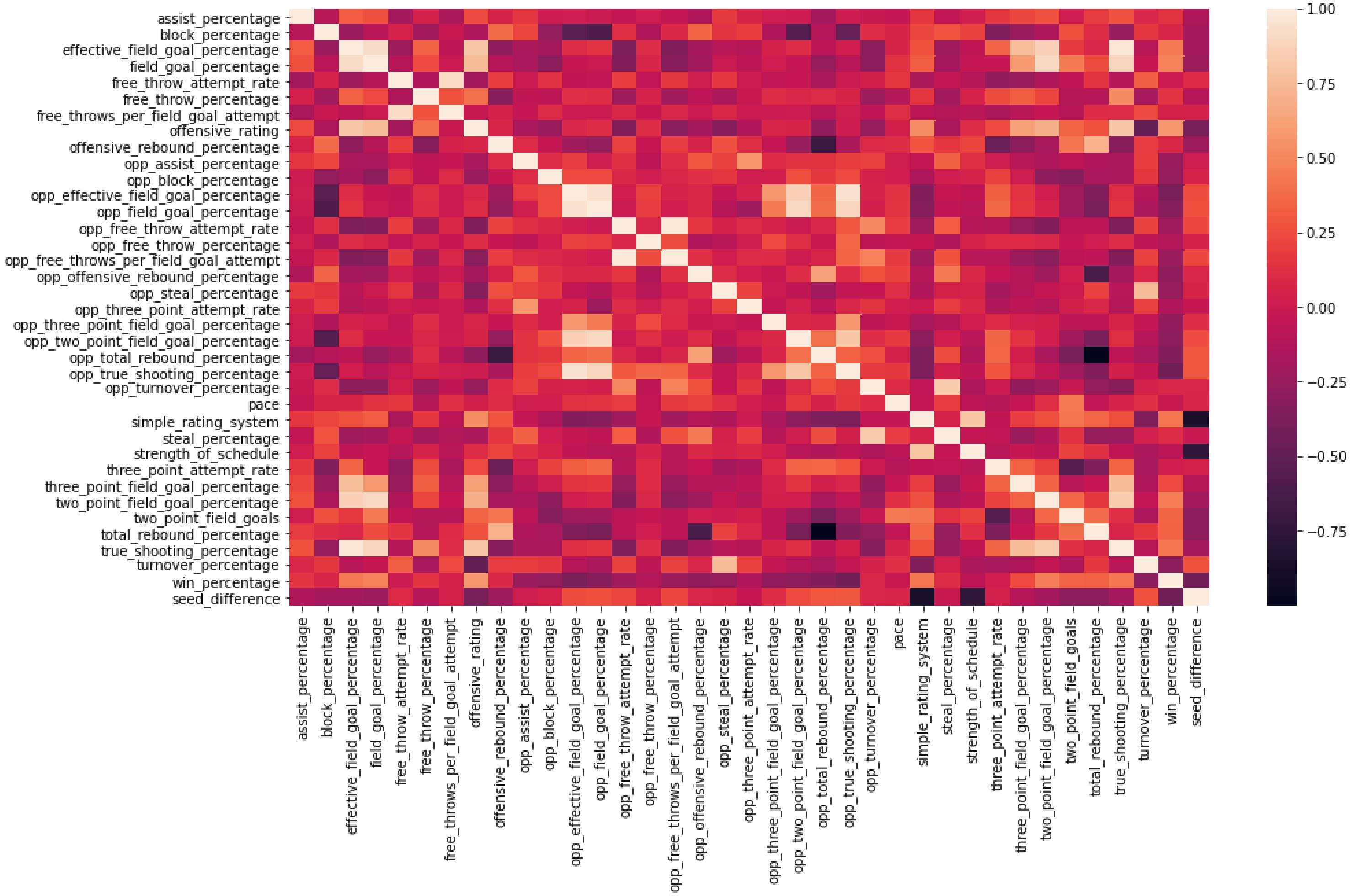
Omnibus:	12.714	**Durbin-Watson:**	2.002
Prob(Omnibus):	0.002	**Jarque-Bera (JB):**	18.351
Skew:	0.176	**Prob(JB):**	0.000104
Kurtosis:	3.758	**Cond. No.**	3.18e+06

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 3.18e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [42]:

```
#Correlation plot involving all current features
plt.figure(figsize = (15,8))
sns.heatmap(features_differentials.corr())
```

Out[42]: <AxesSubplot:>



Focusing on features involving data on offense

In [43]:

```
features_differentials_offense = features_differentials[['assist_percentage',
                                        'effective_field_goal_percentage',
                                        'field_goal_percentage', 'free_throw_attempt_rate', 'free_throw_percentage',
                                        'free_throws_per_field_goal_attempt',
                                        'offensive_rating', 'offensive_rebound_percentage',
                                        'pace', 'three_point_attempt_rate',
                                        'three_point_field_goal_percentage',
                                        'two_point_field_goal_percentage',
                                        'true_shooting_percentage']]
```

```
OLS(response,features_differentials_offense).fit().summary()
```

Out[43]:

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	0.245
Model:	OLS	**Adj. R-squared (uncentered):**	0.229
Method:	Least Squares	**F-statistic:**	15.37
Date:	Sat, 23 Apr 2022	**Prob (F-statistic):**	2.17e-30
Time:	21:30:43	**Log-Likelihood:**	-2490.8
No. Observations:	630	**AIC:**	5008.
Df Residuals:	617	**BIC:**	5065.
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
assist_percentage	-15.8171	7.888	-2.005	0.045	-31.308	-0.326
effective_field_goal_percentage	-579.3802	656.698	-0.882	0.378	-1869.015	710.254
field_goal_percentage	576.9386	639.549	0.902	0.367	-679.017	1832.894
free_throw_attempt_rate	-29.4427	145.340	-0.203	0.840	-314.863	255.978
free_throw_percentage	-3.5894	76.755	-0.047	0.963	-154.323	147.144
free_throws_per_field_goal_attempt	20.3267	222.509	0.091	0.927	-416.640	457.294
offensive_rating	1.1227	0.166	6.762	0.000	0.797	1.449
offensive_rebound_percentage	37.2494	14.215	2.620	0.009	9.333	65.166
pace	-0.3225	0.133	-2.429	0.015	-0.583	-0.062
three_point_attempt_rate	106.9348	112.602	0.950	0.343	-114.195	328.065
three_point_field_goal_percentage	95.8262	99.863	0.960	0.338	-100.286	291.938
two_point_field_goal_percentage	21.2871	149.214	0.143	0.887	-271.742	314.317
true_shooting_percentage	-83.7715	420.809	-0.199	0.842	-910.162	742.619

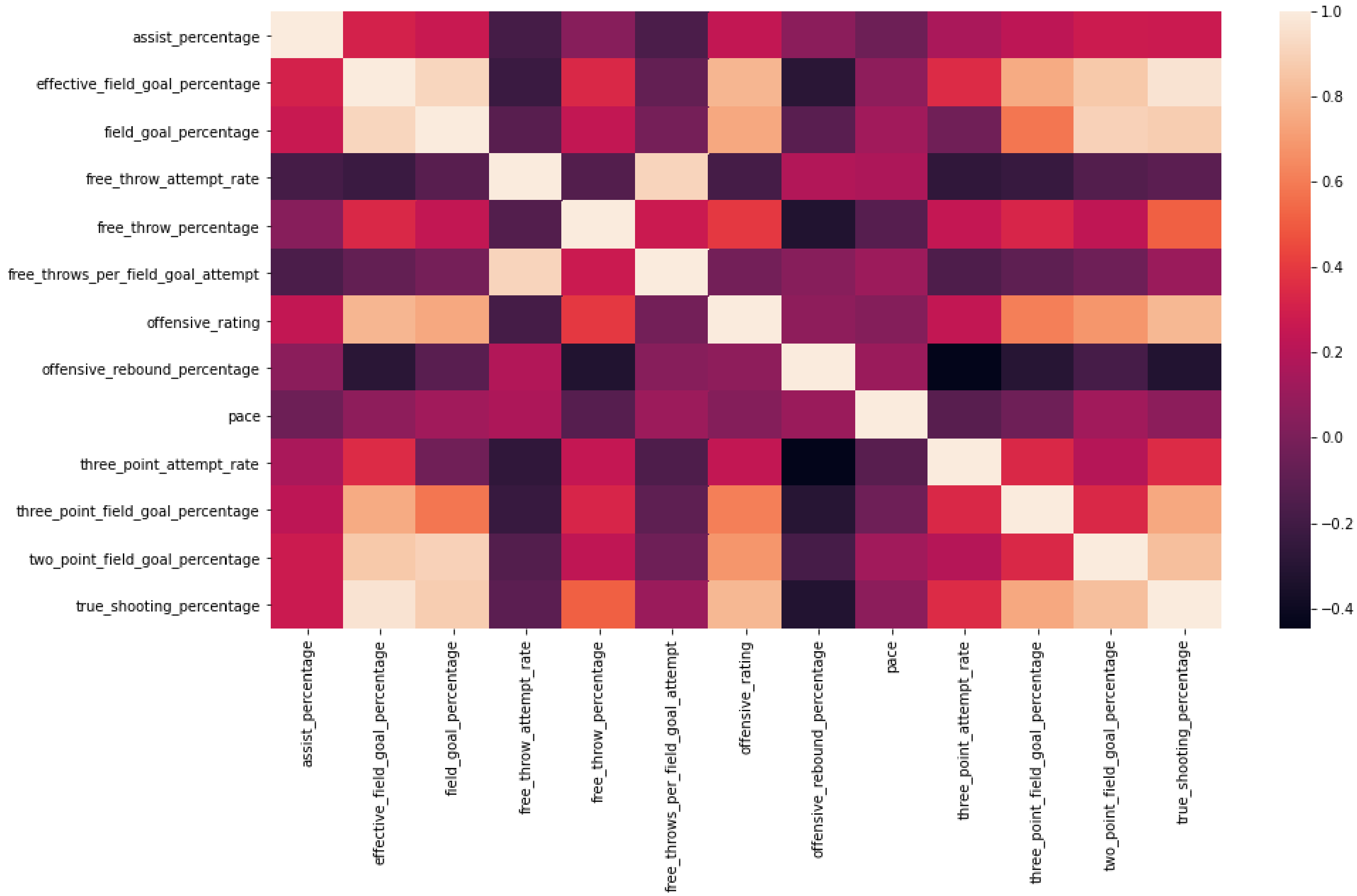
Omnibus:	0.532	**Durbin-Watson:**	1.910
Prob(Omnibus):	0.766	**Jarque-Bera (JB):**	0.389
Skew:	0.040	**Prob(JB):**	0.823
Kurtosis:	3.092	**Cond. No.**	1.26e+04

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[3] The condition number is large, 1.26e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [44]:
```
#Correlation plot involving all offensive features
plt.figure(figsize = (15,8))
sns.heatmap(features_differentials_offense.corr())
```

Out[44]: <AxesSubplot:>



Focusing on features involving data on defense

In [45]:
```
features_differentials_defense = features_differentials[['block_percentage',
                                   'opp_assist_percentage',
                                   'opp_block_percentage',
                                   'opp_effective_field_goal_percentage','opp_field_goal_percentage',
                                   'opp_free_throw_attempt_rate',
                                   'opp_free_throw_percentage',
```

```
                                        'opp_free_throws_per_field_goal_attempt',
                                        'opp_offensive_rebound_percentage',
                                        'opp_steal_percentage',
                                        'opp_three_point_attempt_rate','opp_three_point_field_goal_percentage',
                                        'opp_two_point_field_goal_percentage','opp_total_rebound_percentage',
                                        'opp_true_shooting_percentage', 'opp_turnover_percentage',
                                        'steal_percentage']]
OLS(response,features_differentials_defense).fit().summary()
```

Out[45]:

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	0.259
Model:	OLS	**Adj. R-squared (uncentered):**	0.238
Method:	Least Squares	**F-statistic:**	12.57
Date:	Sat, 23 Apr 2022	**Prob (F-statistic):**	2.25e-30
Time:	21:30:48	**Log-Likelihood:**	-2484.9
No. Observations:	630	**AIC:**	5004.
Df Residuals:	613	**BIC:**	5079.
Df Model:	17		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
block_percentage	23.3786	19.317	1.210	0.227	-14.557	61.314
opp_assist_percentage	-8.6142	10.212	-0.844	0.399	-28.668	11.440
opp_block_percentage	29.5726	27.069	1.092	0.275	-23.587	82.732
opp_effective_field_goal_percentage	-608.2020	661.480	-0.919	0.358	-1907.244	690.840
opp_field_goal_percentage	-431.8069	710.826	-0.607	0.544	-1827.756	964.142
opp_free_throw_attempt_rate	-447.3955	218.971	-2.043	0.041	-877.420	-17.371
opp_free_throw_percentage	-219.4898	100.809	-2.177	0.030	-417.462	-21.517
opp_free_throws_per_field_goal_attempt	510.8092	330.982	1.543	0.123	-139.187	1160.805
opp_offensive_rebound_percentage	39.2037	20.883	1.877	0.061	-1.807	80.214
opp_steal_percentage	-211.8354	38.049	-5.567	0.000	-286.557	-137.113
opp_three_point_attempt_rate	-50.8714	112.898	-0.451	0.652	-272.585	170.842
opp_three_point_field_goal_percentage	194.6839	113.402	1.717	0.087	-28.019	417.387
opp_two_point_field_goal_percentage	382.9942	171.322	2.236	0.026	46.544	719.444
opp_total_rebound_percentage	-148.2268	23.001	-6.444	0.000	-193.396	-103.057
opp_true_shooting_percentage	484.3657	431.214	1.123	0.262	-362.470	1331.201
opp_turnover_percentage	3744.2056	3896.346	0.961	0.337	-3907.601	1.14e+04
steal_percentage	98.4518	44.416	2.217	0.027	11.226	185.677

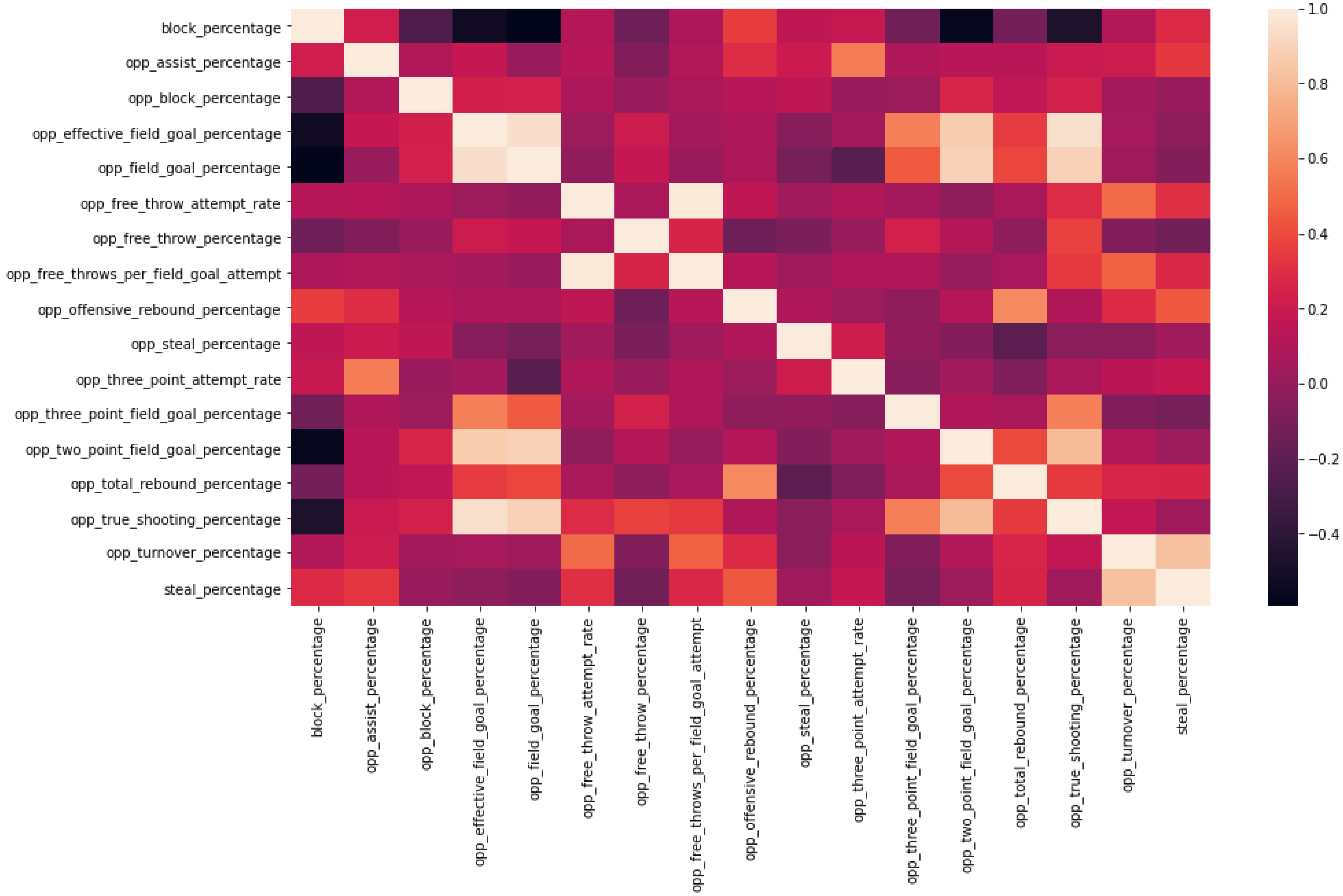
Omnibus:	3.206	**Durbin-Watson:**	1.882

Prob(Omnibus):	0.201	**Jarque-Bera (JB):**	3.026
Skew:	0.140	**Prob(JB):**	0.220
Kurtosis:	3.192	**Cond. No.**	782.

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [46]:
```
#Correlation plot involving all defensive features
plt.figure(figsize = (15,8))
sns.heatmap(features_differentials_defense.corr())
```

Out[46]: <AxesSubplot:>



Feature Selection

- Anything with significance under .5 and most signficant representing groups with high correlation(ex:effective_field_goal_percentage over true_shooting_percentage)
- Removed from defense: 'opp_effective_field_goal_percentage','opp_field_goal_percentage',opp_turnover_percentage,'opp_three_point_attempt_rate',
- Removed from offense: 'effective_field_goal_percentage','field_goal_percentage','free_throw_attempt_rate', 'free_throw_percentage','free_throws_per_field_goal_attempt', 'true_shooting_percentage',
- Keeping 'two_point_field_goal_percentage' from offense to be consistent with defense features ('opp_two_point_field_goal_percentage')

In [47]:
```
features_differentials_final = features_differentials[['assist_percentage', 'block_percentage',
                                    'offensive_rating', 'offensive_rebound_percentage',
                                    'opp_assist_percentage',
                                    'opp_block_percentage',
                                    'opp_free_throw_attempt_rate',
                                    'opp_free_throw_percentage',
                                    'opp_free_throws_per_field_goal_attempt',
```

```
                                             'opp_offensive_rebound_percentage',
                                             'opp_steal_percentage',
                                             'opp_three_point_field_goal_percentage',
                                             'opp_two_point_field_goal_percentage','opp_total_rebound_percentage',
                                             'opp_true_shooting_percentage',
                                             'pace', 'simple_rating_system', 'steal_percentage',
                                             'strength_of_schedule', 'three_point_attempt_rate',
                                             'three_point_field_goal_percentage',
                                             'two_point_field_goal_percentage',
                                             'true_shooting_percentage',
                                             'turnover_percentage','win_percentage','seed_difference']]
features_differentials_final
```

Out[47]:

	assist_percentage	block_percentage	offensive_rating	offensive_rebound_percentage	opp_assist_percentage	opp_block_percentage	opp_free_throw_attempt_rate	opp_free_throw_percentage	opp_
0	-0.089	0.088	7.6	-0.039	0.042	-0.022	0.034	0.014	
1	-0.228	0.104	2.3	0.036	-0.112	-0.017	-0.071	-0.002	
2	0.015	0.019	8.7	-0.058	0.010	-0.025	0.081	0.002	
3	0.020	-0.033	-8.9	-0.099	0.016	0.007	0.006	-0.008	
4	-0.002	-0.067	-3.7	-0.100	-0.010	0.020	-0.123	-0.022	
...	...	...	...	...	...	...	...	...	
625	0.013	-0.004	-4.9	0.020	-0.028	-0.002	0.074	0.040	
626	0.043	0.003	0.5	0.041	0.106	0.012	-0.047	0.045	
627	0.057	-0.035	13.9	-0.065	0.017	-0.001	-0.072	0.000	
628	-0.031	-0.036	-2.3	-0.048	-0.096	0.028	-0.081	-0.034	
629	-0.011	-0.021	15.1	0.029	-0.046	-0.044	-0.024	-0.039	

630 rows × 26 columns

In [48]:

```
#Making sure the team names and years of games are combined with the same team statistic differentials before train/test split
pre_training_features = pd.concat([features_differentials_final,game_info], axis=1)
pre_training_features
```

Out[48]:

	assist_percentage	block_percentage	offensive_rating	offensive_rebound_percentage	opp_assist_percentage	opp_block_percentage	opp_free_throw_attempt_rate	opp_free_throw_percentage	opp_
0	-0.089	0.088	7.6	-0.039	0.042	-0.022	0.034	0.014	
1	-0.228	0.104	2.3	0.036	-0.112	-0.017	-0.071	-0.002	
2	0.015	0.019	8.7	-0.058	0.010	-0.025	0.081	0.002	
3	0.020	-0.033	-8.9	-0.099	0.016	0.007	0.006	-0.008	
4	-0.002	-0.067	-3.7	-0.100	-0.010	0.020	-0.123	-0.022	
...	...	...	...	...	...	...	...	...	

	assist_percentage	block_percentage	offensive_rating	offensive_rebound_percentage	opp_assist_percentage	opp_block_percentage	opp_free_throw_attempt_rate	opp_free_throw_percentage	opp_
625	0.013	-0.004	-4.9	0.020	-0.028	-0.002	0.074	0.040	
626	0.043	0.003	0.5	0.041	0.106	0.012	-0.047	0.045	
627	0.057	-0.035	13.9	-0.065	0.017	-0.001	-0.072	0.000	
628	-0.031	-0.036	-2.3	-0.048	-0.096	0.028	-0.081	-0.034	
629	-0.011	-0.021	15.1	0.029	-0.046	-0.044	-0.024	-0.039	

630 rows × 29 columns

Create training set

In [49]:

```
#Create training(75%) and test(25%) data
X_train, X_test, y_train, y_test = train_test_split(pre_training_features,response,random_state=42)

game_info_train = X_train[['Year','Home_Team','Away_Team']]
game_info_test = X_test[['Year','Home_Team','Away_Team']]

#This data below will not be needed in model making but used later for result analysis using the above game_info DataFrames
X_train=X_train.drop(columns=['Year','Home_Team','Away_Team'])
X_test=X_test.drop(columns=['Year','Home_Team','Away_Team'])
```

Scale the data

In [50]:

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Creating benchmark feature sets

Better Seed Benchmark

In [51]:

```
#Only input feature is seed differential, with the assumption that better seed will win in each prediction
X_train_BetterSeed = X_train[['seed_difference']]
X_train_BetterSeed_Scaled = scaler.fit_transform(X_train_BetterSeed)

X_test_BetterSeed = X_test[['seed_difference']]
X_test_BetterSeed_Scaled = scaler.transform(X_test_BetterSeed)
```

Better Record Benchmark

In [52]:
```
#Similar to seed_differential benchmark, but teams with worse seeds can still have better records than teams with better seeds
X_train_BetterRecord = X_train[['win_percentage']]
X_train_BetterRecord_Scaled = scaler.fit_transform(X_train_BetterRecord)

X_test_BetterRecord = X_test[['win_percentage']]
X_test_BetterRecord_Scaled = scaler.transform(X_test_BetterRecord)
```

Using PCA (Scaled & Not-Scaled)

Not scaled

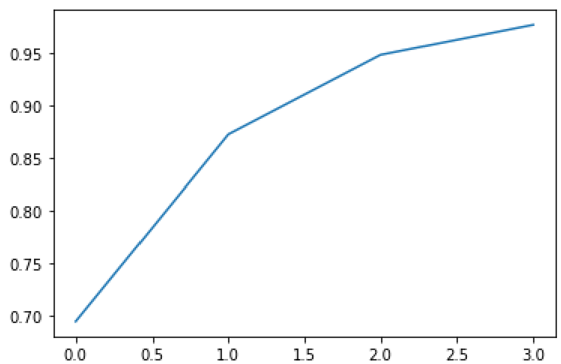
In [53]:
```
pca = PCA(.95)
pca.fit(X_train)
pca.n_components_
```

Out[53]: 4

In [54]:
```
X_train_PCA = pca.transform(X_train)
X_test_PCA = pca.transform(X_test)
```

In [55]:
```
plt.plot(pca.explained_variance_ratio_.cumsum())
```

Out[55]: [<matplotlib.lines.Line2D at 0x1b5f2a40190>]



Scaled

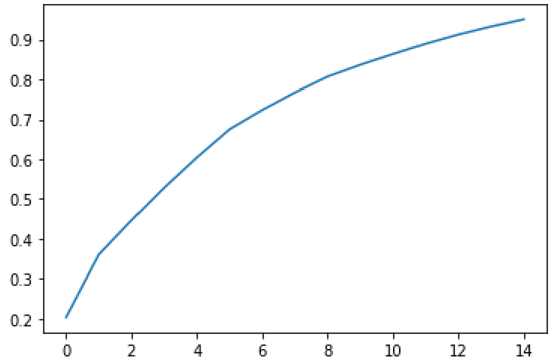
In [56]:
```
pca_scaled = PCA(.95)
pca_scaled.fit(X_train_scaled)
pca_scaled.n_components_
```

Out[56]: 15

In [57]:
```
X_train_scaled_PCA = pca_scaled.transform(X_train_scaled)
X_test_scaled_PCA = pca_scaled.transform(X_test_scaled)
```

In [58]:
```
plt.plot(pca_scaled.explained_variance_ratio_.cumsum())
```

Out[58]: [<matplotlib.lines.Line2D at 0x1b5f2dd35b0>]



Run benchmark models using Linear Regression

Better Seed Benchmark Model

In [59]:
```
#Not scaled BetterSeed Model, find MSE
BetterSeed_BenchmarkModel = LinearRegression()

BetterSeed_BenchmarkModel.fit(X_train_BetterSeed, y_train)

y_pred_BetterSeed = BetterSeed_BenchmarkModel.predict(X_test_BetterSeed)

mse_BetterSeed = mean_squared_error(y_test, y_pred_BetterSeed)

#Scaled BetterSeed Model, find MSE
BetterSeed_BenchmarkModel_Scaled = LinearRegression()

BetterSeed_BenchmarkModel_Scaled.fit(X_train_BetterSeed_Scaled, y_train)

y_pred_BetterSeed_Scaled = BetterSeed_BenchmarkModel_Scaled.predict(X_test_BetterSeed_Scaled)

mse_BetterSeed_Scaled = mean_squared_error(y_test, y_pred_BetterSeed_Scaled)
```

Better Record Benchmark Model

In [60]:
```
#Not scaled BetterRecord Model, find MSE
BetterRecord_BenchmarkModel = LinearRegression()

BetterRecord_BenchmarkModel.fit(X_train_BetterRecord, y_train)

y_pred_BetterRecord = BetterSeed_BenchmarkModel.predict(X_test_BetterRecord)

mse_BetterRecord = mean_squared_error(y_test, y_pred_BetterRecord)
```

```
#Scaled BetterRecord Model, find MSE
BetterRecord_BenchmarkModel_Scaled = LinearRegression()

BetterRecord_BenchmarkModel_Scaled.fit(X_train_BetterRecord_Scaled, y_train)

y_pred_BetterRecord_Scaled = BetterRecord_BenchmarkModel_Scaled.predict(X_test_BetterRecord_Scaled)

mse_BetterRecord_Scaled = mean_squared_error(y_test, y_pred_BetterRecord_Scaled)
```

Perform Grid Searches

Perform Individual Model Grid Searches

GradientBoostingRegressor GridSearch

In [61]:

```
# -----
# Coarse-Grained GradientBoostingRegressor GridSearch
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'n_estimators': list(range(100, 1000, 100)),'learning_rate': [.01,.1,.2,.3,.4,.5,.6,.7,.8,.9,1]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 792 candidates, totalling 2376 fits
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 600}
```

In [60]:

```
# -----
# Refined GradientBoostingRegressor GridSearch
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [1,2], 'n_estimators': list(range(500, 700, 50)),'learning_rate': [.005,.01,.015]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 550}
```

In [38]:

```
# -----
# Final GradientBoostingRegressor GridSearch
#   Repeated search to make sure that optimal hyperparameter values found are indeed optimal
# -----

param_grid={'max_depth': [1,2], 'n_estimators': list(range(500, 600, 25)),'learning_rate': [.005,.01,.015]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)
```

```
grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 550}
```

On this dataset, the optimal model parameters for the `GradientBoostingRegressor` class are:

- `learning_rate = 0.01`
- `max_depth = 1`
- `n_estimators = 550`

GradientBoostingRegressor GridSearch w/Scaling

In [32]:

```
# -----
# Coarse-Grained GradientBoostingRegressor GridSearch with Scaled features
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'n_estimators': list(range(100, 1000, 100)),'learning_rate': [.01,.1,.2,.3,.4,.5,.6,.7,.8,.9,1]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 792 candidates, totalling 2376 fits
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 600}
```

In [33]:

```
# -----
# Refined GradientBoostingRegressor GridSearch with Scaled features
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [1,2], 'n_estimators': list(range(500, 700, 50)),'learning_rate': [.005,.01,.015]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 600}
```

In [34]:

```
# -----
# Final GradientBoostingRegressor GridSearch with Scaled features
#   Repeated search to make sure that optimal hyperparameter values found are indeed optimal
# -----

param_grid={'max_depth': [1,2], 'n_estimators': list(range(550, 650, 25)),'learning_rate': [.005,.01,.015]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

```
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 600}
```

On this dataset, the optimal model parameters for the `GradientBoostingRegressor` class w/Scaling are:

- `learning_rate = 0.01`
- `max_depth = 1`
- `n_estimators = 600`

GradientBoostingRegressor GridSearch PCA

In [32]:

```
# -----
# Coarse-Grained GradientBoostingRegressor GridSearch with PCA
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'n_estimators': list(range(100, 1000, 100)),'learning_rate': [.01,.1,.2,.3,.4,.5,.6,.7,.8,.9,1]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 792 candidates, totalling 2376 fits
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 600}
```

In [33]:

```
# -----
# Refined GradientBoostingRegressor GridSearch with PCA
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [1,2], 'n_estimators': list(range(500, 700, 50)),'learning_rate': [.005,.01,.015]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 550}
```

In [34]:

```
# -----
# Final GradientBoostingRegressor GridSearch with PCA
#   Repeated search to make sure that optimal hyperparameter values found are indeed optimal
# -----

param_grid={'max_depth': [1,2], 'n_estimators': list(range(500, 600, 25)),'learning_rate': [.005,.01,.015]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
The best parameters are:  {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 550}
```

On this dataset, the optimal model parameters for the `GradientBoostingRegressor` class PCA are:

- `learning_rate = 0.01`

- `max_depth = 1`
- `n_estimators = 550`

GradientBoostingRegressor GridSearch PCA w/Scaling

In [35]:

```
# -----
# Coarse-Grained GradientBoostingRegressor GridSearch with PCA & Scaled features
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'n_estimators': list(range(100, 1000, 100)),'learning_rate': [.01,.1,.2,.3,.4,.5,.6,.7,.8,.9,1]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 792 candidates, totalling 2376 fits
The best parameters are:  {'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 100}
```

In [36]:

```
# -----
# Refined GradientBoostingRegressor GridSearch with PCA & Scaled features
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [1,2], 'n_estimators': list(range(50, 150, 50)),'learning_rate': [.05,.1,.15]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
The best parameters are:  {'learning_rate': 0.15, 'max_depth': 1, 'n_estimators': 100}
```

In [37]:

```
# -----
# Final GradientBoostingRegressor GridSearch with PCA & Scaled features
#   Repeated search to make sure that optimal hyperparameter values found are indeed optimal
# -----

param_grid={'max_depth': [1,2], 'n_estimators': list(range(75, 125, 25)),'learning_rate': [.125,.15,.175]}
classifier = GradientBoostingRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
The best parameters are:  {'learning_rate': 0.15, 'max_depth': 1, 'n_estimators': 100}
```

On this dataset, the optimal model parameters for the `GradientBoostingRegressor` class PCA w/Scaling are:

- `learning_rate = 0.15`
- `max_depth = 1`
- `n_estimators = 100`

RandomForestRegressor GridSearch

In [62]:
```
# -----
# Coarse-Grained RandomForestRegressor GridSearch
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'n_estimators': list(range(100, 1000, 100)),'min_samples_split': list(range(2, 20, 3))}
classifier = RandomForestRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 432 candidates, totalling 1296 fits
The best parameters are:  {'max_depth': 4, 'min_samples_split': 5, 'n_estimators': 200}
```

In [37]:
```
# -----
# Refined RandomForestRegressor GridSearch
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [3,4,5], 'n_estimators': list(range(100, 300, 50)),'min_samples_split': list(range(2, 3, 1))}
classifier = RandomForestRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
The best parameters are:  {'max_depth': 4, 'min_samples_split': 2, 'n_estimators': 200}
```

On this dataset, the optimal model parameters for the `RandomForestRegressor` class are:

- `max_depth = 4`
- `n_estimators = 200`
- `min_samples_split = 2`

RandomForestRegressor GridSearch w/Scaling

In [51]:
```
# -----
# Coarse-Grained RandomForestRegressor GridSearch with Scaled features
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'n_estimators': list(range(100, 1000, 100)),'min_samples_split': list(range(2, 20, 3))}
classifier = RandomForestRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 432 candidates, totalling 1296 fits
The best parameters are:  {'max_depth': 4, 'min_samples_split': 5, 'n_estimators': 200}
```

In [39]:
```
# -----
# Refined RandomForestRegressor GridSearch with Scaled features
```

```
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [3,4,5], 'n_estimators': list(range(100, 300, 50)),'min_samples_split': list(range(4, 6, 1))}
classifier = RandomForestRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
The best parameters are:  {'max_depth': 4, 'min_samples_split': 5, 'n_estimators': 200}
```

On this dataset, the optimal model parameters for the `RandomForestRegressor` class w/Scaling are:

- `max_depth = 4`
- `n_estimators = 200`
- `min_samples_split = 5`

RandomForestRegressor GridSearch PCA

In [38]:

```
# -----
# Coarse-Grained RandomForestRegressor GridSearch with PCA
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'n_estimators': list(range(100, 1000, 100)),'min_samples_split': list(range(2, 20, 3))}
classifier = RandomForestRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 432 candidates, totalling 1296 fits
The best parameters are:  {'max_depth': 2, 'min_samples_split': 2, 'n_estimators': 100}
```

In [40]:

```
# -----
# Refined RandomForestRegressor GridSearch with PCA
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [1,2,3], 'n_estimators': list(range(50, 150, 50)),'min_samples_split': list(range(2, 3, 1))}
classifier = RandomForestRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
The best parameters are:  {'max_depth': 2, 'min_samples_split': 2, 'n_estimators': 100}
```

On this dataset, the optimal model parameters for the `RandomForestRegressor` class PCA are:

- `max_depth = 2`
- `n_estimators = 100`
- `min_samples_split = 2`

RandomForestRegressor GridSearch PCA w/Scaling

In [41]:
```
# -----
# Coarse-Grained RandomForestRegressor GridSearch with PCA & Scaled features
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'n_estimators': list(range(100, 1000, 100)),'min_samples_split': list(range(2, 20, 3))}
classifier = RandomForestRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 432 candidates, totalling 1296 fits
The best parameters are:  {'max_depth': 8, 'min_samples_split': 5, 'n_estimators': 900}
```

In [42]:
```
# -----
# Refined RandomForestRegressor GridSearch with PCA & Scaled features
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [7,8,9], 'n_estimators': list(range(850, 950, 50)),'min_samples_split': list(range(4, 6, 1))}
classifier = RandomForestRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
The best parameters are:  {'max_depth': 7, 'min_samples_split': 5, 'n_estimators': 900}
```

On this dataset, the optimal model parameters for the `RandomForestRegressor` class PCA w/Scaling are:

- `max_depth = 7`
- `n_estimators = 900`
- `min_samples_split = 5`

DecisionTreeRegressor GridSearch

In [43]:
```
# -----
# Coarse-Grained DecisionTreeRegressor GridSearch
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'splitter' : ["best", "random"],'min_samples_split': list(range(2, 20, 3))}
classifier = DecisionTreeRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 96 candidates, totalling 288 fits
The best parameters are:  {'max_depth': 3, 'min_samples_split': 14, 'splitter': 'best'}
```

In [44]:
```
# -----
# Refined DecisionTreeRegressor GridSearch
```

```
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [2,3,4], 'splitter' : ["best", "random"],'min_samples_split': list(range(13, 15, 1))}
classifier = DecisionTreeRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
The best parameters are:  {'max_depth': 3, 'min_samples_split': 13, 'splitter': 'best'}
```

On this dataset, the optimal model parameters for the `DecisionTreeRegressor` class are:

- `splitter = 'best`
- `max_depth = 3`
- `min_samples_split = 13`

DecisionTreeRegressor GridSearch w/Scaling

In [37]:

```
# -----
# Coarse-Grained DecisionTreeRegressor GridSearch with Scaled features
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'splitter' : ["best", "random"],'min_samples_split': list(range(2, 20, 3))}
classifier = DecisionTreeRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 96 candidates, totalling 288 fits
The best parameters are:  {'max_depth': 3, 'min_samples_split': 14, 'splitter': 'best'}
```

In [38]:

```
# -----
# Refined DecisionTreeRegressor GridSearch with Scaled features
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [2,3,4], 'splitter' : ["best", "random"],'min_samples_split': list(range(13, 15, 1))}
classifier = DecisionTreeRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
The best parameters are:  {'max_depth': 3, 'min_samples_split': 13, 'splitter': 'best'}
```

On this dataset, the optimal model parameters for the `DecisionTreeRegressor` class w/Scaling are:

- `splitter = 'best`
- `max_depth = 3`
- `min_samples_split = 13`

DecisionTreeRegressor GridSearch PCA

In [47]:
```
# -----
# Coarse-Grained DecisionTreeRegressor GridSearch with PCA features
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'splitter' : ["best", "random"],'min_samples_split': list(range(2, 20, 3))}
classifier = DecisionTreeRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 96 candidates, totalling 288 fits
The best parameters are:  {'max_depth': 5, 'min_samples_split': 17, 'splitter': 'random'}
```

In [48]:
```
# -----
# Refined DecisionTreeRegressor GridSearch with PCA features
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [5,6,7], 'splitter' : ["best", "random"],'min_samples_split': list(range(15, 20, 1))}
classifier = DecisionTreeRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
The best parameters are:  {'max_depth': 6, 'min_samples_split': 18, 'splitter': 'random'}
```

On this dataset, the optimal model parameters for the `DecisionTreeRegressor` class PCA are:

- `splitter = 'random`
- `max_depth = 6`
- `min_samples_split = 18`

DecisionTreeRegressor GridSearch PCA w/Scaling

In [49]:
```
# -----
# Coarse-Grained DecisionTreeRegressor GridSearch with PCA & Scaled features
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'max_depth': [1,2,3,4,5,8,16,32], 'splitter' : ["best", "random"],'min_samples_split': list(range(2, 20, 3))}
classifier = DecisionTreeRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 96 candidates, totalling 288 fits
The best parameters are:  {'max_depth': 3, 'min_samples_split': 2, 'splitter': 'random'}
```

In [40]:
```
# -----
# Refined DecisionTreeRegressor GridSearch with PCA & Scaled features
```

```
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'max_depth': [2,3,4], 'splitter' : ["best", "random"],'min_samples_split': list(range(2, 3, 1))}
classifier = DecisionTreeRegressor(random_state=42)
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled_PCA, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
The best parameters are:  {'max_depth': 3, 'min_samples_split': 2, 'splitter': 'random'}
```

On this dataset, the optimal model parameters for the `DecisionTreeRegressor` class PCA w/Scaling are:

- `splitter = 'random`
- `max_depth = 3`
- `min_samples_split = 2`

K-Neighbors GridSearch

In [97]:

```
# -----
# Coarse-Grained KNeighborsRegressor GridSearch
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'n_neighbors': [11,21,31,41,51], 'weights' : ["uniform", "distance"],'metric': ["euclidean", "manhattan"]}
classifier = KNeighborsRegressor()
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
The best parameters are:  {'metric': 'euclidean', 'n_neighbors': 21, 'weights': 'distance'}
```

In [99]:

```
# -----
# Refined KNeighborsRegressor GridSearch
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'n_neighbors': [17,19,21,23], 'weights' : ["uniform", "distance"],'metric': ["euclidean", "manhattan"]}
classifier = KNeighborsRegressor()
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
The best parameters are:  {'metric': 'manhattan', 'n_neighbors': 19, 'weights': 'distance'}
```

On this dataset, the optimal model parameters for the `KNeighborsRegressor` class are:

- `metric = 'manhattan`
- `n_neighbors = 19`
- `weights = 'distance`

K-Neighbors Scaled GridSearch

In [100…

```
# -----
# Coarse-Grained KNeighborsRegressor GridSearch with scaled features
#   Wide search based on hyperparameter ranges provided
# -----

param_grid={'n_neighbors': [11,21,31,41,51], 'weights' : ["uniform", "distance"],'metric': ["euclidean", "manhattan"]}
classifier = KNeighborsRegressor()
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
The best parameters are:  {'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}
```

In [41]:

```
# -----
# Refined KNeighborsRegressor GridSearch with scaled features
#   Smaller window search based on optimal hyperparameter values found in initial broad search
# -----

param_grid={'n_neighbors': [19,11,13], 'weights' : ["uniform", "distance"],'metric': ["euclidean", "manhattan"]}
classifier = KNeighborsRegressor()
grid_search_cv = GridSearchCV(estimator = classifier,param_grid=param_grid,verbose=1,cv=3)

grid_search_cv.fit(X_train_scaled, y_train)
print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
The best parameters are:  {'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}
```

On this dataset, the optimal model parameters for the `KNeighborsRegressor` class are:

- `metric = 'manhattan`
- `n_neighbors = 11`
- `weights = 'distance`

Fit Optimal Models

Using the optimal parameters found in the GridSearch for each model

GradientBoostingRegressor

In [62]:

```
gbrt = GradientBoostingRegressor(max_depth=1,n_estimators =550,learning_rate=.01,random_state=42)
gbrt.fit(X_train,y_train)
```

Out[62]:

```
GradientBoostingRegressor(learning_rate=0.01, max_depth=1, n_estimators=550,
                          random_state=42)
```

GradientBoostingRegressor w/Scaling

In [63]:

```
gbrt_scaled = GradientBoostingRegressor(max_depth=1,n_estimators =600,learning_rate=.01,random_state=42)
```

```
gbrt_scaled.fit(X_train_scaled,y_train)
```

Out[63]:
```
GradientBoostingRegressor(learning_rate=0.01, max_depth=1, n_estimators=600,
                          random_state=42)
```

GradientBoostingRegressor PCA

In [64]:
```
gbrt_PCA = GradientBoostingRegressor(max_depth=1,n_estimators =550,learning_rate=.01,random_state=42)
gbrt_PCA.fit(X_train_PCA,y_train)
```

Out[64]:
```
GradientBoostingRegressor(learning_rate=0.01, max_depth=1, n_estimators=550,
                          random_state=42)
```

GradientBoostingRegressor PCA w/Scaling

In [65]:
```
gbrt_scaled_PCA = GradientBoostingRegressor(max_depth=1,n_estimators =100,learning_rate=.15,random_state=42)
gbrt_scaled_PCA.fit(X_train_scaled_PCA,y_train)
```

Out[65]:
```
GradientBoostingRegressor(learning_rate=0.15, max_depth=1, random_state=42)
```

RandomForestRegressor

In [66]:
```
rnd_reg = RandomForestRegressor(max_depth=4,n_estimators =200,min_samples_split=2,random_state=42)
rnd_reg.fit(X_train,y_train)
```

Out[66]:
```
RandomForestRegressor(max_depth=4, n_estimators=200, random_state=42)
```

RandomForestRegressor w/Scaling

In [67]:
```
rnd_reg_scaled = RandomForestRegressor(max_depth=4,n_estimators =200,min_samples_split=5,random_state=42)
rnd_reg_scaled.fit(X_train_scaled,y_train)
```

Out[67]:
```
RandomForestRegressor(max_depth=4, min_samples_split=5, n_estimators=200,
                      random_state=42)
```

RandomForestRegressor PCA

In [68]:
```
rnd_reg_PCA = RandomForestRegressor(max_depth=2,n_estimators =100,min_samples_split=2,random_state=42)
rnd_reg_PCA.fit(X_train_PCA,y_train)
```

Out[68]:
```
RandomForestRegressor(max_depth=2, random_state=42)
```

RandomForestRegressor PCA w/Scaling

In [69]:
```
rnd_reg_scaled_PCA = RandomForestRegressor(max_depth=7,n_estimators =900,min_samples_split=5,random_state=42)
rnd_reg_scaled_PCA.fit(X_train_scaled_PCA,y_train)
```

Out[69]:
```
RandomForestRegressor(max_depth=7, min_samples_split=5, n_estimators=900,
                      random_state=42)
```

DecisionTreeRegressor

In [70]:
```
tree_reg = DecisionTreeRegressor(splitter='best', max_depth=3, min_samples_split=13, random_state=42)
tree_reg.fit(X_train,y_train)
```

Out[70]:
```
DecisionTreeRegressor(max_depth=3, min_samples_split=13, random_state=42)
```

DecisionTreeRegressor w/Scaling

In [71]:
```
tree_reg_scaled = DecisionTreeRegressor(splitter='best', max_depth=3, min_samples_split=13, random_state=42)
tree_reg_scaled.fit(X_train_scaled,y_train)
```

Out[71]:
```
DecisionTreeRegressor(max_depth=3, min_samples_split=13, random_state=42)
```

DecisionTreeRegressor PCA

In [72]:
```
tree_reg_PCA = DecisionTreeRegressor(splitter='random', max_depth=6, min_samples_split=18, random_state=42)
tree_reg_PCA.fit(X_train_PCA,y_train)
```

Out[72]:
```
DecisionTreeRegressor(max_depth=6, min_samples_split=18, random_state=42,
                      splitter='random')
```

DecisionTreeRegressor PCA w/Scaling

In [73]:
```
tree_reg_scaled_PCA = DecisionTreeRegressor(splitter='random', max_depth=3, min_samples_split=2, random_state=42)
tree_reg_scaled_PCA.fit(X_train_scaled_PCA,y_train)
```

Out[73]:
```
DecisionTreeRegressor(max_depth=3, random_state=42, splitter='random')
```

KNeighborsRegressor

In [74]:
```
knn_reg = KNeighborsRegressor(metric=  'manhattan', n_neighbors= 19, weights= 'distance')
knn_reg.fit(X_train,y_train)
```

Out[74]:
```
KNeighborsRegressor(metric='manhattan', n_neighbors=19, weights='distance')
```

KNeighborsRegressor w/Scaling

In [75]:
```
knn_scaled_reg = KNeighborsRegressor(metric=  'manhattan', n_neighbors= 11, weights= 'distance')
knn_scaled_reg.fit(X_train,y_train)
```

Out[75]:
```
KNeighborsRegressor(metric='manhattan', n_neighbors=11, weights='distance')
```

VotingRegressor

Creating a model combining the Gradient Boosting, Random Forest, Decision Tree, and K-Nearest Neighbor Models.

In [76]:
```
voting_reg = VotingRegressor(
                        [('gbrt',gbrt),
                          ('rf', rnd_reg),
                          ('tree', tree_reg),
                          ('knn', knn_reg)])

voting_reg.fit(X_train,y_train)
```

Out[76]:
```
VotingRegressor(estimators=[('gbrt',
                             GradientBoostingRegressor(learning_rate=0.01,
                                                       max_depth=1,
                                                       n_estimators=550,
                                                       random_state=42)),
                            ('rf',
                             RandomForestRegressor(max_depth=4,
                                                   n_estimators=200,
                                                   random_state=42)),
                            ('tree',
                             DecisionTreeRegressor(max_depth=3,
                                                   min_samples_split=13,
                                                   random_state=42)),
                            ('knn',
                             KNeighborsRegressor(metric='manhattan',
                                                 n_neighbors=19,
                                                 weights='distance'))])
```

Artificial Neural Networks

In [77]:
```
#Split training data into smaller training set and validation set
X_train_ANN, X_valid_ANN, y_train_ANN, y_valid_ANN = train_test_split(X_train,y_train,test_size=0.25,random_state=42)
```

In [78]:
```
#Check the shape of the training sets for input shape parameter
X_train_ANN.shape
#x_test.shape
```

Out[78]:
```
(354, 26)
```

In [79]:
```
#Run Sequential Class model with 4 layers

tf.random.set_seed(42)

model_ANN = keras.models.Sequential()
model_ANN.add(keras.layers.Flatten(input_shape=[26]))
model_ANN.add(keras.layers.Dense(1000,activation="softmax"))
model_ANN.add(keras.layers.Dense(100,activation="softmax"))
model_ANN.add(keras.layers.Dense(10,activation="softmax"))
model_ANN.add(keras.layers.Dense(1))


model_ANN.compile(loss="mean_squared_error",
              optimizer=keras.optimizers.SGD(lr=0.01))
```

```
history = model_ANN.fit(X_train_ANN,y_train_ANN,epochs=1000,validation_data=(X_valid_ANN,y_valid_ANN))
```

```
C:\Users\gfann\anaconda3\lib\site-packages\keras\optimizer_v2\gradient_descent.py:102: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
Epoch 1/1000
12/12 [==============================] - 1s 19ms/step - loss: 199.6507 - val_loss: 234.4536
Epoch 2/1000
12/12 [==============================] - 0s 4ms/step - loss: 189.4930 - val_loss: 224.9577
Epoch 3/1000
12/12 [==============================] - 0s 4ms/step - loss: 183.9006 - val_loss: 219.9217
Epoch 4/1000
12/12 [==============================] - 0s 4ms/step - loss: 181.3486 - val_loss: 216.7481
Epoch 5/1000
12/12 [==============================] - 0s 6ms/step - loss: 179.7345 - val_loss: 215.3087
Epoch 6/1000
12/12 [==============================] - 0s 4ms/step - loss: 179.0559 - val_loss: 213.8610
Epoch 7/1000
12/12 [==============================] - 0s 6ms/step - loss: 178.5149 - val_loss: 213.4208
Epoch 8/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.4557 - val_loss: 212.6750
Epoch 9/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1884 - val_loss: 210.7381
Epoch 10/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7979 - val_loss: 210.3898
Epoch 11/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7830 - val_loss: 210.8348
Epoch 12/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7493 - val_loss: 211.0012
Epoch 13/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8650 - val_loss: 210.5993
Epoch 14/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7411 - val_loss: 211.6289
Epoch 15/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8831 - val_loss: 211.1710
Epoch 16/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8800 - val_loss: 210.9820
Epoch 17/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7895 - val_loss: 210.3751
Epoch 18/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7624 - val_loss: 211.4923
Epoch 19/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8348 - val_loss: 211.4922
Epoch 20/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9154 - val_loss: 212.1659
Epoch 21/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9611 - val_loss: 210.7790
Epoch 22/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7660 - val_loss: 210.5660
Epoch 23/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7658 - val_loss: 210.9018
Epoch 24/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7683 - val_loss: 211.3637
Epoch 25/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7814 - val_loss: 211.6585
Epoch 26/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9797 - val_loss: 212.2142
Epoch 27/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0627 - val_loss: 210.7958
Epoch 28/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8182 - val_loss: 210.7889
Epoch 29/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7972 - val_loss: 210.1451
Epoch 30/1000
```

```
12/12 [==============================] - 0s 5ms/step - loss: 177.8132 - val_loss: 210.1129
Epoch 31/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7698 - val_loss: 210.1625
Epoch 32/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8163 - val_loss: 209.9174
Epoch 33/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8885 - val_loss: 210.0219
Epoch 34/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7752 - val_loss: 210.5354
Epoch 35/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7422 - val_loss: 210.4316
Epoch 36/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8366 - val_loss: 210.7680
Epoch 37/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7571 - val_loss: 210.4383
Epoch 38/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7701 - val_loss: 209.9561
Epoch 39/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.9270 - val_loss: 210.3510
Epoch 40/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7818 - val_loss: 209.7026
Epoch 41/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9712 - val_loss: 209.6924
Epoch 42/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9126 - val_loss: 209.9263
Epoch 43/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8099 - val_loss: 210.3468
Epoch 44/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8011 - val_loss: 210.2717
Epoch 45/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8231 - val_loss: 209.8918
Epoch 46/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8162 - val_loss: 210.9130
Epoch 47/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.9261 - val_loss: 210.9550
Epoch 48/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8478 - val_loss: 210.3604
Epoch 49/1000
12/12 [==============================] - 0s 8ms/step - loss: 177.7831 - val_loss: 210.6887
Epoch 50/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7603 - val_loss: 211.3136
Epoch 51/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8636 - val_loss: 210.5065
Epoch 52/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8013 - val_loss: 210.7050
Epoch 53/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9287 - val_loss: 210.1547
Epoch 54/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8382 - val_loss: 211.1520
Epoch 55/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8235 - val_loss: 211.2191
Epoch 56/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.9608 - val_loss: 210.4143
Epoch 57/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8413 - val_loss: 210.1878
Epoch 58/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8159 - val_loss: 209.9789
Epoch 59/1000
12/12 [==============================] - 0s 10ms/step - loss: 177.8190 - val_loss: 210.7602
Epoch 60/1000
12/12 [==============================] - 0s 12ms/step - loss: 177.7793 - val_loss: 210.6309
Epoch 61/1000
12/12 [==============================] - 0s 7ms/step - loss: 177.7859 - val_loss: 210.7114
Epoch 62/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7966 - val_loss: 210.8536
```

```
Epoch 63/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7605 - val_loss: 210.3947
Epoch 64/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8263 - val_loss: 210.6804
Epoch 65/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7560 - val_loss: 211.1401
Epoch 66/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8698 - val_loss: 211.2651
Epoch 67/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9142 - val_loss: 210.9746
Epoch 68/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7822 - val_loss: 209.9559
Epoch 69/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7888 - val_loss: 209.4060
Epoch 70/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.2153 - val_loss: 209.2898
Epoch 71/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1848 - val_loss: 209.0963
Epoch 72/1000
12/12 [==============================] - 0s 6ms/step - loss: 178.7404 - val_loss: 209.2455
Epoch 73/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.2371 - val_loss: 209.4025
Epoch 74/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.2192 - val_loss: 209.3546
Epoch 75/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1706 - val_loss: 209.3163
Epoch 76/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.2552 - val_loss: 209.3812
Epoch 77/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0928 - val_loss: 209.3871
Epoch 78/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1347 - val_loss: 209.1783
Epoch 79/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.4680 - val_loss: 209.7047
Epoch 80/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9008 - val_loss: 209.3737
Epoch 81/1000
12/12 [==============================] - 0s 6ms/step - loss: 178.1540 - val_loss: 210.0319
Epoch 82/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8376 - val_loss: 210.0926
Epoch 83/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8176 - val_loss: 210.6437
Epoch 84/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8800 - val_loss: 210.5288
Epoch 85/1000
12/12 [==============================] - 0s 8ms/step - loss: 177.7997 - val_loss: 210.6513
Epoch 86/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7234 - val_loss: 211.5889
Epoch 87/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8765 - val_loss: 211.1316
Epoch 88/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8359 - val_loss: 210.7274
Epoch 89/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7411 - val_loss: 211.4178
Epoch 90/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8034 - val_loss: 210.9102
Epoch 91/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9232 - val_loss: 209.5842
Epoch 92/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0569 - val_loss: 209.1708
Epoch 93/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.4587 - val_loss: 209.1914
Epoch 94/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.4456 - val_loss: 209.6518
Epoch 95/1000
```

```
12/12 [==============================] - 0s 5ms/step - loss: 177.8890 - val_loss: 209.5500
Epoch 96/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9704 - val_loss: 209.4803
Epoch 97/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0820 - val_loss: 210.2516
Epoch 98/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7274 - val_loss: 209.9256
Epoch 99/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8080 - val_loss: 209.4900
Epoch 100/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9949 - val_loss: 209.5809
Epoch 101/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9898 - val_loss: 209.8756
Epoch 102/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8338 - val_loss: 210.2037
Epoch 103/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7991 - val_loss: 209.7931
Epoch 104/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8393 - val_loss: 209.4218
Epoch 105/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.2232 - val_loss: 209.4022
Epoch 106/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.2172 - val_loss: 209.4544
Epoch 107/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1064 - val_loss: 209.9597
Epoch 108/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8376 - val_loss: 210.4776
Epoch 109/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8302 - val_loss: 210.5603
Epoch 110/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7668 - val_loss: 210.0854
Epoch 111/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8635 - val_loss: 210.3640
Epoch 112/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7689 - val_loss: 210.1933
Epoch 113/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8260 - val_loss: 209.8089
Epoch 114/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8892 - val_loss: 209.4118
Epoch 115/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.2160 - val_loss: 209.3546
Epoch 116/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1382 - val_loss: 209.6875
Epoch 117/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8931 - val_loss: 209.7206
Epoch 118/1000
12/12 [==============================] - 0s 6ms/step - loss: 178.0092 - val_loss: 209.4227
Epoch 119/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0477 - val_loss: 209.3324
Epoch 120/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.2714 - val_loss: 209.4453
Epoch 121/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1025 - val_loss: 209.5043
Epoch 122/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0545 - val_loss: 209.8587
Epoch 123/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8156 - val_loss: 210.5294
Epoch 124/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7364 - val_loss: 211.3783
Epoch 125/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7853 - val_loss: 211.6799
Epoch 126/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9841 - val_loss: 211.2343
Epoch 127/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7999 - val_loss: 210.6849
```

```
Epoch 128/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7940 - val_loss: 210.2012
Epoch 129/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8149 - val_loss: 210.1166
Epoch 130/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7934 - val_loss: 210.1250
Epoch 131/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8330 - val_loss: 210.4530
Epoch 132/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7847 - val_loss: 209.7579
Epoch 133/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9167 - val_loss: 210.3621
Epoch 134/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8318 - val_loss: 210.4037
Epoch 135/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7441 - val_loss: 210.3949
Epoch 136/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7829 - val_loss: 211.0062
Epoch 137/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9266 - val_loss: 210.8748
Epoch 138/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7829 - val_loss: 210.9666
Epoch 139/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7885 - val_loss: 210.7783
Epoch 140/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8080 - val_loss: 210.4034
Epoch 141/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7885 - val_loss: 210.5509
Epoch 142/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8603 - val_loss: 211.6326
Epoch 143/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8967 - val_loss: 211.0633
Epoch 144/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8065 - val_loss: 211.1607
Epoch 145/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8199 - val_loss: 210.6939
Epoch 146/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7554 - val_loss: 210.3714
Epoch 147/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7654 - val_loss: 210.5888
Epoch 148/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7232 - val_loss: 211.0045
Epoch 149/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7754 - val_loss: 210.7925
Epoch 150/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8236 - val_loss: 210.7319
Epoch 151/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8465 - val_loss: 210.5902
Epoch 152/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8235 - val_loss: 210.6774
Epoch 153/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7741 - val_loss: 212.1446
Epoch 154/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9911 - val_loss: 211.2832
Epoch 155/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8473 - val_loss: 210.6538
Epoch 156/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7558 - val_loss: 210.6682
Epoch 157/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7748 - val_loss: 210.5817
Epoch 158/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7760 - val_loss: 211.1675
Epoch 159/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8706 - val_loss: 210.7736
Epoch 160/1000
```

```
12/12 [==============================] - 0s 4ms/step - loss: 177.8452 - val_loss: 211.8754
Epoch 161/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9541 - val_loss: 211.0072
Epoch 162/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7747 - val_loss: 211.6254
Epoch 163/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9426 - val_loss: 211.1214
Epoch 164/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8138 - val_loss: 210.5322
Epoch 165/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7616 - val_loss: 210.1759
Epoch 166/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8274 - val_loss: 210.6963
Epoch 167/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7309 - val_loss: 210.4198
Epoch 168/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7524 - val_loss: 210.5226
Epoch 169/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7654 - val_loss: 210.8576
Epoch 170/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8124 - val_loss: 210.3932
Epoch 171/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7403 - val_loss: 210.5336
Epoch 172/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7948 - val_loss: 210.6777
Epoch 173/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8731 - val_loss: 210.1003
Epoch 174/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8626 - val_loss: 210.2000
Epoch 175/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9103 - val_loss: 210.2719
Epoch 176/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7646 - val_loss: 210.4724
Epoch 177/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8294 - val_loss: 210.2377
Epoch 178/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8351 - val_loss: 210.5787
Epoch 179/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8792 - val_loss: 210.4532
Epoch 180/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8206 - val_loss: 210.5071
Epoch 181/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8736 - val_loss: 210.7024
Epoch 182/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7789 - val_loss: 211.1748
Epoch 183/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8599 - val_loss: 211.7821
Epoch 184/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9307 - val_loss: 210.9924
Epoch 185/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7951 - val_loss: 210.7097
Epoch 186/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8466 - val_loss: 210.0049
Epoch 187/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7784 - val_loss: 210.7799
Epoch 188/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7503 - val_loss: 210.8375
Epoch 189/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8235 - val_loss: 211.0683
Epoch 190/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8795 - val_loss: 210.1658
Epoch 191/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8204 - val_loss: 210.4155
Epoch 192/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7382 - val_loss: 211.0031
```

```
Epoch 193/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7433 - val_loss: 210.9844
Epoch 194/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8104 - val_loss: 210.8822
Epoch 195/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8555 - val_loss: 210.3899
Epoch 196/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8248 - val_loss: 211.7286
Epoch 197/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8931 - val_loss: 211.6031
Epoch 198/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8363 - val_loss: 210.9301
Epoch 199/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7882 - val_loss: 210.6426
Epoch 200/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7863 - val_loss: 210.9243
Epoch 201/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7940 - val_loss: 210.9165
Epoch 202/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7619 - val_loss: 210.7580
Epoch 203/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8687 - val_loss: 210.2535
Epoch 204/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8031 - val_loss: 211.2295
Epoch 205/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8563 - val_loss: 210.9382
Epoch 206/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7577 - val_loss: 210.2234
Epoch 207/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7883 - val_loss: 209.9366
Epoch 208/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9596 - val_loss: 210.4751
Epoch 209/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8196 - val_loss: 209.9665
Epoch 210/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7843 - val_loss: 210.5447
Epoch 211/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7816 - val_loss: 210.6906
Epoch 212/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8357 - val_loss: 209.9611
Epoch 213/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8907 - val_loss: 210.2432
Epoch 214/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7669 - val_loss: 210.5435
Epoch 215/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7839 - val_loss: 210.8629
Epoch 216/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7621 - val_loss: 210.5715
Epoch 217/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7105 - val_loss: 210.3780
Epoch 218/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8231 - val_loss: 211.2006
Epoch 219/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7805 - val_loss: 210.2679
Epoch 220/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8397 - val_loss: 210.4506
Epoch 221/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7255 - val_loss: 210.6096
Epoch 222/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7841 - val_loss: 211.3259
Epoch 223/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8708 - val_loss: 210.6059
Epoch 224/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8503 - val_loss: 210.1716
Epoch 225/1000
```

```
12/12 [==============================] - 0s 4ms/step - loss: 177.8217 - val_loss: 210.0689
Epoch 226/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8123 - val_loss: 210.8511
Epoch 227/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7524 - val_loss: 210.0266
Epoch 228/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7967 - val_loss: 209.8500
Epoch 229/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8582 - val_loss: 210.0442
Epoch 230/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8940 - val_loss: 210.3848
Epoch 231/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7722 - val_loss: 210.0503
Epoch 232/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8990 - val_loss: 209.7477
Epoch 233/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8921 - val_loss: 209.9556
Epoch 234/1000
12/12 [==============================] - 0s 3ms/step - loss: 177.8300 - val_loss: 210.3548
Epoch 235/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8602 - val_loss: 209.9223
Epoch 236/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9062 - val_loss: 210.0230
Epoch 237/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8343 - val_loss: 209.8701
Epoch 238/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9237 - val_loss: 210.3979
Epoch 239/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7501 - val_loss: 209.5974
Epoch 240/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0525 - val_loss: 209.9315
Epoch 241/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8235 - val_loss: 210.3154
Epoch 242/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7419 - val_loss: 210.7653
Epoch 243/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9004 - val_loss: 211.0092
Epoch 244/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7350 - val_loss: 211.8938
Epoch 245/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9638 - val_loss: 211.2986
Epoch 246/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7836 - val_loss: 212.2098
Epoch 247/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0593 - val_loss: 212.3204
Epoch 248/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0043 - val_loss: 211.2662
Epoch 249/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9763 - val_loss: 210.7743
Epoch 250/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7987 - val_loss: 211.1494
Epoch 251/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7981 - val_loss: 210.6215
Epoch 252/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7663 - val_loss: 210.7041
Epoch 253/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7941 - val_loss: 211.0658
Epoch 254/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7935 - val_loss: 212.5249
Epoch 255/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1288 - val_loss: 212.9100
Epoch 256/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1958 - val_loss: 211.8106
Epoch 257/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0142 - val_loss: 212.2079
```

```
Epoch 258/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0208 - val_loss: 212.3100
Epoch 259/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0052 - val_loss: 211.0817
Epoch 260/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9031 - val_loss: 211.0021
Epoch 261/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8405 - val_loss: 210.5900
Epoch 262/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7218 - val_loss: 210.9235
Epoch 263/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8471 - val_loss: 209.8141
Epoch 264/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9348 - val_loss: 210.0031
Epoch 265/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8471 - val_loss: 209.9728
Epoch 266/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8310 - val_loss: 210.4526
Epoch 267/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8631 - val_loss: 211.4426
Epoch 268/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8310 - val_loss: 211.6371
Epoch 269/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9613 - val_loss: 210.6634
Epoch 270/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7409 - val_loss: 210.8232
Epoch 271/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7579 - val_loss: 210.6594
Epoch 272/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8095 - val_loss: 210.3576
Epoch 273/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7635 - val_loss: 210.6596
Epoch 274/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8460 - val_loss: 209.9432
Epoch 275/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8962 - val_loss: 209.7564
Epoch 276/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9236 - val_loss: 210.5780
Epoch 277/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7972 - val_loss: 210.7326
Epoch 278/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7675 - val_loss: 210.4955
Epoch 279/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7209 - val_loss: 210.2824
Epoch 280/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7815 - val_loss: 210.6383
Epoch 281/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7709 - val_loss: 210.3447
Epoch 282/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8017 - val_loss: 209.9133
Epoch 283/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8055 - val_loss: 209.9521
Epoch 284/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8911 - val_loss: 210.4566
Epoch 285/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7228 - val_loss: 210.7790
Epoch 286/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7881 - val_loss: 211.0462
Epoch 287/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7743 - val_loss: 211.4342
Epoch 288/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8588 - val_loss: 211.3145
Epoch 289/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8134 - val_loss: 210.9858
Epoch 290/1000
```

```
12/12 [==============================] - 0s 4ms/step - loss: 177.8177 - val_loss: 211.2444
Epoch 291/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8443 - val_loss: 211.3340
Epoch 292/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8906 - val_loss: 210.6938
Epoch 293/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7579 - val_loss: 210.4167
Epoch 294/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9059 - val_loss: 209.9140
Epoch 295/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8953 - val_loss: 210.0406
Epoch 296/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7639 - val_loss: 210.3672
Epoch 297/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8051 - val_loss: 210.5120
Epoch 298/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7140 - val_loss: 210.3109
Epoch 299/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8343 - val_loss: 210.0884
Epoch 300/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8640 - val_loss: 209.6714
Epoch 301/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0039 - val_loss: 211.1264
Epoch 302/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8464 - val_loss: 210.6454
Epoch 303/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8819 - val_loss: 210.1189
Epoch 304/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7981 - val_loss: 210.8805
Epoch 305/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8153 - val_loss: 210.3113
Epoch 306/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7932 - val_loss: 211.4435
Epoch 307/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8373 - val_loss: 210.7920
Epoch 308/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7636 - val_loss: 210.4203
Epoch 309/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7676 - val_loss: 211.5063
Epoch 310/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8533 - val_loss: 210.9882
Epoch 311/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7679 - val_loss: 210.7597
Epoch 312/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9552 - val_loss: 211.2025
Epoch 313/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8324 - val_loss: 211.2888
Epoch 314/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8597 - val_loss: 209.9848
Epoch 315/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8934 - val_loss: 209.5610
Epoch 316/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9738 - val_loss: 209.8087
Epoch 317/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9112 - val_loss: 209.4704
Epoch 318/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0913 - val_loss: 209.5714
Epoch 319/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0316 - val_loss: 209.4586
Epoch 320/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1264 - val_loss: 209.9242
Epoch 321/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9868 - val_loss: 209.5094
Epoch 322/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1046 - val_loss: 210.3139
```

```
Epoch 323/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8031 - val_loss: 209.8039
Epoch 324/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8668 - val_loss: 210.0690
Epoch 325/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7915 - val_loss: 210.9054
Epoch 326/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8328 - val_loss: 211.9618
Epoch 327/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.9550 - val_loss: 212.0118
Epoch 328/1000
12/12 [==============================] - 0s 3ms/step - loss: 177.9177 - val_loss: 211.3845
Epoch 329/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8524 - val_loss: 211.0305
Epoch 330/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8240 - val_loss: 210.5123
Epoch 331/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8302 - val_loss: 210.4018
Epoch 332/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7656 - val_loss: 210.3965
Epoch 333/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7616 - val_loss: 210.2717
Epoch 334/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7405 - val_loss: 210.6483
Epoch 335/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7524 - val_loss: 210.7195
Epoch 336/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7985 - val_loss: 210.5869
Epoch 337/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8067 - val_loss: 210.4182
Epoch 338/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7403 - val_loss: 210.6400
Epoch 339/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8019 - val_loss: 211.0614
Epoch 340/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9397 - val_loss: 211.1530
Epoch 341/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8150 - val_loss: 210.7155
Epoch 342/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7831 - val_loss: 210.3920
Epoch 343/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7191 - val_loss: 211.0199
Epoch 344/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8054 - val_loss: 210.3525
Epoch 345/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7733 - val_loss: 210.2847
Epoch 346/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8540 - val_loss: 210.6124
Epoch 347/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7726 - val_loss: 210.6533
Epoch 348/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7483 - val_loss: 210.2646
Epoch 349/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8028 - val_loss: 209.7186
Epoch 350/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9257 - val_loss: 210.3184
Epoch 351/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7774 - val_loss: 210.3470
Epoch 352/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8173 - val_loss: 211.3850
Epoch 353/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8544 - val_loss: 211.5925
Epoch 354/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8914 - val_loss: 211.0265
Epoch 355/1000
```

```
12/12 [==============================] - 0s 5ms/step - loss: 177.7655 - val_loss: 212.0805
Epoch 356/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9374 - val_loss: 211.1994
Epoch 357/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7576 - val_loss: 210.9464
Epoch 358/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8448 - val_loss: 210.8895
Epoch 359/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7645 - val_loss: 211.0714
Epoch 360/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7776 - val_loss: 212.1886
Epoch 361/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0130 - val_loss: 212.3462
Epoch 362/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0316 - val_loss: 211.7615
Epoch 363/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9260 - val_loss: 211.2659
Epoch 364/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8082 - val_loss: 210.3043
Epoch 365/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7909 - val_loss: 210.4671
Epoch 366/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7064 - val_loss: 211.2865
Epoch 367/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8141 - val_loss: 211.5183
Epoch 368/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8708 - val_loss: 211.2488
Epoch 369/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9246 - val_loss: 211.4640
Epoch 370/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9122 - val_loss: 213.4758
Epoch 371/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.3719 - val_loss: 212.0776
Epoch 372/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9705 - val_loss: 211.2117
Epoch 373/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8689 - val_loss: 210.8448
Epoch 374/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8020 - val_loss: 211.3777
Epoch 375/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8802 - val_loss: 211.2132
Epoch 376/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8357 - val_loss: 210.6287
Epoch 377/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7753 - val_loss: 210.0819
Epoch 378/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8061 - val_loss: 210.3160
Epoch 379/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8329 - val_loss: 210.3040
Epoch 380/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7632 - val_loss: 209.9693
Epoch 381/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8850 - val_loss: 209.5615
Epoch 382/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0023 - val_loss: 209.8042
Epoch 383/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9428 - val_loss: 209.2849
Epoch 384/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.2391 - val_loss: 209.5316
Epoch 385/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0030 - val_loss: 209.8653
Epoch 386/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8724 - val_loss: 209.9288
Epoch 387/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8211 - val_loss: 210.2352
```

```
Epoch 388/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8686 - val_loss: 209.8695
Epoch 389/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8487 - val_loss: 209.5237
Epoch 390/1000
12/12 [==============================] - 0s 6ms/step - loss: 178.0659 - val_loss: 209.4970
Epoch 391/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0325 - val_loss: 209.5463
Epoch 392/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1016 - val_loss: 209.9797
Epoch 393/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8367 - val_loss: 210.0038
Epoch 394/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8761 - val_loss: 209.6272
Epoch 395/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9809 - val_loss: 209.8705
Epoch 396/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8902 - val_loss: 210.4689
Epoch 397/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8072 - val_loss: 210.4246
Epoch 398/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7965 - val_loss: 210.1086
Epoch 399/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8067 - val_loss: 209.9326
Epoch 400/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9287 - val_loss: 210.1526
Epoch 401/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7262 - val_loss: 209.8987
Epoch 402/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8504 - val_loss: 210.0258
Epoch 403/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9051 - val_loss: 209.9303
Epoch 404/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9269 - val_loss: 209.3137
Epoch 405/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1878 - val_loss: 209.1567
Epoch 406/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.5318 - val_loss: 209.1198
Epoch 407/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.6072 - val_loss: 209.2753
Epoch 408/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.2891 - val_loss: 209.2949
Epoch 409/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.2759 - val_loss: 210.0075
Epoch 410/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8141 - val_loss: 210.0085
Epoch 411/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8306 - val_loss: 210.8960
Epoch 412/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7725 - val_loss: 210.6704
Epoch 413/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7483 - val_loss: 210.3753
Epoch 414/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8432 - val_loss: 209.9406
Epoch 415/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8704 - val_loss: 210.1803
Epoch 416/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8566 - val_loss: 210.0091
Epoch 417/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8161 - val_loss: 210.6072
Epoch 418/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8431 - val_loss: 211.3335
Epoch 419/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8333 - val_loss: 210.5680
Epoch 420/1000
```

```
12/12 [==============================] - 0s 4ms/step - loss: 177.7528 - val_loss: 210.0276
Epoch 421/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8080 - val_loss: 210.7051
Epoch 422/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8130 - val_loss: 209.8868
Epoch 423/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9395 - val_loss: 210.0119
Epoch 424/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8655 - val_loss: 210.0137
Epoch 425/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9951 - val_loss: 209.7148
Epoch 426/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8793 - val_loss: 209.7919
Epoch 427/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9938 - val_loss: 210.1982
Epoch 428/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7853 - val_loss: 209.8194
Epoch 429/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8908 - val_loss: 209.9043
Epoch 430/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7933 - val_loss: 209.8614
Epoch 431/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8269 - val_loss: 209.8387
Epoch 432/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8848 - val_loss: 210.2215
Epoch 433/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7641 - val_loss: 211.2200
Epoch 434/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8135 - val_loss: 211.3578
Epoch 435/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8487 - val_loss: 210.5684
Epoch 436/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7685 - val_loss: 210.5490
Epoch 437/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7883 - val_loss: 210.6147
Epoch 438/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8056 - val_loss: 210.3999
Epoch 439/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7970 - val_loss: 209.9240
Epoch 440/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8375 - val_loss: 209.8156
Epoch 441/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8603 - val_loss: 210.2949
Epoch 442/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7704 - val_loss: 210.8156
Epoch 443/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8502 - val_loss: 210.8383
Epoch 444/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7701 - val_loss: 210.6638
Epoch 445/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7848 - val_loss: 210.4349
Epoch 446/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7619 - val_loss: 210.1501
Epoch 447/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7892 - val_loss: 209.6680
Epoch 448/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9524 - val_loss: 209.7956
Epoch 449/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9042 - val_loss: 209.9560
Epoch 450/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8029 - val_loss: 210.1230
Epoch 451/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7880 - val_loss: 210.3427
Epoch 452/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7917 - val_loss: 210.2553
```

```
Epoch 453/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7653 - val_loss: 210.3487
Epoch 454/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7321 - val_loss: 210.1891
Epoch 455/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8251 - val_loss: 210.2409
Epoch 456/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8142 - val_loss: 210.3487
Epoch 457/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7372 - val_loss: 210.0059
Epoch 458/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8919 - val_loss: 209.7333
Epoch 459/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8775 - val_loss: 209.9007
Epoch 460/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8293 - val_loss: 210.1362
Epoch 461/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7771 - val_loss: 210.5299
Epoch 462/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7939 - val_loss: 210.0368
Epoch 463/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8636 - val_loss: 210.4320
Epoch 464/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8065 - val_loss: 210.5009
Epoch 465/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8335 - val_loss: 210.9588
Epoch 466/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7523 - val_loss: 210.5424
Epoch 467/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7863 - val_loss: 210.1134
Epoch 468/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7645 - val_loss: 210.2557
Epoch 469/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7083 - val_loss: 210.3223
Epoch 470/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7579 - val_loss: 210.9438
Epoch 471/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8305 - val_loss: 210.8546
Epoch 472/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7832 - val_loss: 210.4518
Epoch 473/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7841 - val_loss: 210.3258
Epoch 474/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8278 - val_loss: 211.0931
Epoch 475/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7905 - val_loss: 211.3582
Epoch 476/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8232 - val_loss: 210.7744
Epoch 477/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7378 - val_loss: 209.8581
Epoch 478/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9708 - val_loss: 210.7469
Epoch 479/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8828 - val_loss: 211.0555
Epoch 480/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8075 - val_loss: 211.6663
Epoch 481/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8989 - val_loss: 211.3415
Epoch 482/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8630 - val_loss: 211.4408
Epoch 483/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8333 - val_loss: 210.9924
Epoch 484/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7504 - val_loss: 211.6365
Epoch 485/1000
```

```
12/12 [==============================] - 0s 4ms/step - loss: 177.8752 - val_loss: 211.6351
Epoch 486/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8529 - val_loss: 210.7496
Epoch 487/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8740 - val_loss: 211.5971
Epoch 488/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8913 - val_loss: 212.2958
Epoch 489/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0677 - val_loss: 211.9288
Epoch 490/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9702 - val_loss: 210.7732
Epoch 491/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8038 - val_loss: 211.8584
Epoch 492/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0277 - val_loss: 211.2226
Epoch 493/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8339 - val_loss: 210.3365
Epoch 494/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7326 - val_loss: 210.7775
Epoch 495/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7997 - val_loss: 209.9939
Epoch 496/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9094 - val_loss: 210.1898
Epoch 497/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9453 - val_loss: 211.4079
Epoch 498/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8160 - val_loss: 211.7271
Epoch 499/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8774 - val_loss: 211.8742
Epoch 500/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9594 - val_loss: 212.0338
Epoch 501/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1299 - val_loss: 210.8556
Epoch 502/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7862 - val_loss: 211.0503
Epoch 503/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7667 - val_loss: 210.2375
Epoch 504/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7900 - val_loss: 209.9330
Epoch 505/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9489 - val_loss: 209.3173
Epoch 506/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.3402 - val_loss: 209.4640
Epoch 507/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1056 - val_loss: 209.8053
Epoch 508/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9099 - val_loss: 209.7365
Epoch 509/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9251 - val_loss: 209.8488
Epoch 510/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8369 - val_loss: 209.5462
Epoch 511/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1018 - val_loss: 209.9597
Epoch 512/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7960 - val_loss: 210.3284
Epoch 513/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8521 - val_loss: 209.9461
Epoch 514/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7802 - val_loss: 210.2570
Epoch 515/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9052 - val_loss: 210.9275
Epoch 516/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7238 - val_loss: 210.8576
Epoch 517/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7539 - val_loss: 211.0748
```

```
Epoch 518/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7981 - val_loss: 210.7817
Epoch 519/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7575 - val_loss: 210.6251
Epoch 520/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7430 - val_loss: 211.0685
Epoch 521/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7592 - val_loss: 210.7178
Epoch 522/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8460 - val_loss: 210.4040
Epoch 523/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7633 - val_loss: 210.6041
Epoch 524/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7594 - val_loss: 210.8316
Epoch 525/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7922 - val_loss: 210.7794
Epoch 526/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7930 - val_loss: 211.1200
Epoch 527/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8218 - val_loss: 211.4335
Epoch 528/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8396 - val_loss: 211.3438
Epoch 529/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9580 - val_loss: 211.0086
Epoch 530/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8270 - val_loss: 211.2676
Epoch 531/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8315 - val_loss: 212.4622
Epoch 532/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1137 - val_loss: 211.2269
Epoch 533/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8625 - val_loss: 210.5668
Epoch 534/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7185 - val_loss: 211.0300
Epoch 535/1000
12/12 [==============================] - 0s 7ms/step - loss: 177.8054 - val_loss: 210.2994
Epoch 536/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7924 - val_loss: 210.4381
Epoch 537/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7891 - val_loss: 210.0631
Epoch 538/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7883 - val_loss: 210.2210
Epoch 539/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7697 - val_loss: 210.8622
Epoch 540/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8075 - val_loss: 211.8126
Epoch 541/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9089 - val_loss: 210.1744
Epoch 542/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8126 - val_loss: 210.7652
Epoch 543/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7925 - val_loss: 211.9243
Epoch 544/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9712 - val_loss: 210.8018
Epoch 545/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8822 - val_loss: 210.7249
Epoch 546/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8000 - val_loss: 210.9953
Epoch 547/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7919 - val_loss: 210.8777
Epoch 548/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7918 - val_loss: 210.7948
Epoch 549/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7222 - val_loss: 210.3197
Epoch 550/1000
```

```
12/12 [==============================] - 0s 5ms/step - loss: 177.9074 - val_loss: 210.1234
Epoch 551/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7775 - val_loss: 211.0194
Epoch 552/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7964 - val_loss: 210.0965
Epoch 553/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8105 - val_loss: 210.6663
Epoch 554/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7371 - val_loss: 210.6364
Epoch 555/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7312 - val_loss: 210.0912
Epoch 556/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7588 - val_loss: 210.2016
Epoch 557/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7506 - val_loss: 210.9139
Epoch 558/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8484 - val_loss: 211.3101
Epoch 559/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8996 - val_loss: 210.8042
Epoch 560/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7932 - val_loss: 210.9856
Epoch 561/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8173 - val_loss: 210.7909
Epoch 562/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7539 - val_loss: 211.4214
Epoch 563/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8474 - val_loss: 210.7424
Epoch 564/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7495 - val_loss: 210.4693
Epoch 565/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8048 - val_loss: 211.0084
Epoch 566/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7786 - val_loss: 211.9680
Epoch 567/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9541 - val_loss: 211.6343
Epoch 568/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0047 - val_loss: 211.1287
Epoch 569/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8183 - val_loss: 211.3661
Epoch 570/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8133 - val_loss: 210.0417
Epoch 571/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7999 - val_loss: 210.4756
Epoch 572/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7872 - val_loss: 211.2267
Epoch 573/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8718 - val_loss: 210.7712
Epoch 574/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7719 - val_loss: 210.3685
Epoch 575/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7929 - val_loss: 210.5180
Epoch 576/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7882 - val_loss: 211.1596
Epoch 577/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9295 - val_loss: 211.0640
Epoch 578/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8379 - val_loss: 210.7240
Epoch 579/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7947 - val_loss: 210.6220
Epoch 580/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7941 - val_loss: 210.2643
Epoch 581/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7566 - val_loss: 210.2009
Epoch 582/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8628 - val_loss: 209.8359
```

```
Epoch 583/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8275 - val_loss: 209.8019
Epoch 584/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8717 - val_loss: 209.8875
Epoch 585/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8234 - val_loss: 209.8979
Epoch 586/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8434 - val_loss: 209.9436
Epoch 587/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8379 - val_loss: 210.1818
Epoch 588/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7706 - val_loss: 210.5837
Epoch 589/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7949 - val_loss: 210.5656
Epoch 590/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7655 - val_loss: 210.9929
Epoch 591/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8650 - val_loss: 210.8697
Epoch 592/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7518 - val_loss: 210.4321
Epoch 593/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7802 - val_loss: 210.2147
Epoch 594/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7910 - val_loss: 210.9587
Epoch 595/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7785 - val_loss: 210.4470
Epoch 596/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7428 - val_loss: 210.2401
Epoch 597/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7767 - val_loss: 211.0708
Epoch 598/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8122 - val_loss: 211.2607
Epoch 599/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9052 - val_loss: 211.6026
Epoch 600/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9258 - val_loss: 211.7274
Epoch 601/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9313 - val_loss: 211.3376
Epoch 602/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7946 - val_loss: 211.2997
Epoch 603/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8710 - val_loss: 211.4454
Epoch 604/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9317 - val_loss: 210.9515
Epoch 605/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8168 - val_loss: 210.7751
Epoch 606/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7401 - val_loss: 210.9019
Epoch 607/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7927 - val_loss: 210.7191
Epoch 608/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8170 - val_loss: 210.5484
Epoch 609/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8478 - val_loss: 211.4787
Epoch 610/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9107 - val_loss: 210.2704
Epoch 611/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8616 - val_loss: 209.6759
Epoch 612/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9667 - val_loss: 209.7853
Epoch 613/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9346 - val_loss: 209.5976
Epoch 614/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9509 - val_loss: 209.5023
Epoch 615/1000
```

```
12/12 [==============================] - 0s 6ms/step - loss: 178.0419 - val_loss: 209.4740
Epoch 616/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0726 - val_loss: 209.9862
Epoch 617/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8298 - val_loss: 210.9628
Epoch 618/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8941 - val_loss: 209.8539
Epoch 619/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8695 - val_loss: 209.7068
Epoch 620/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1007 - val_loss: 209.6154
Epoch 621/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0279 - val_loss: 209.3814
Epoch 622/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1305 - val_loss: 209.6205
Epoch 623/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0210 - val_loss: 209.7085
Epoch 624/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8954 - val_loss: 210.0434
Epoch 625/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8465 - val_loss: 209.7258
Epoch 626/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9574 - val_loss: 209.9611
Epoch 627/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8953 - val_loss: 210.3274
Epoch 628/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7791 - val_loss: 210.3299
Epoch 629/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7436 - val_loss: 211.5364
Epoch 630/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8884 - val_loss: 211.4331
Epoch 631/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8063 - val_loss: 212.0702
Epoch 632/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9814 - val_loss: 211.9475
Epoch 633/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9748 - val_loss: 212.3634
Epoch 634/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0807 - val_loss: 212.4426
Epoch 635/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9878 - val_loss: 212.8228
Epoch 636/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1409 - val_loss: 213.0229
Epoch 637/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.2455 - val_loss: 211.2780
Epoch 638/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8569 - val_loss: 210.8178
Epoch 639/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7365 - val_loss: 210.7370
Epoch 640/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8135 - val_loss: 212.1359
Epoch 641/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0210 - val_loss: 211.8019
Epoch 642/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8764 - val_loss: 211.0519
Epoch 643/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7926 - val_loss: 210.7668
Epoch 644/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7442 - val_loss: 210.3194
Epoch 645/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8500 - val_loss: 209.8243
Epoch 646/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9211 - val_loss: 209.4562
Epoch 647/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0349 - val_loss: 210.0900
```

```
Epoch 648/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8472 - val_loss: 211.0254
Epoch 649/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7638 - val_loss: 211.1665
Epoch 650/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8503 - val_loss: 211.2223
Epoch 651/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8551 - val_loss: 210.8462
Epoch 652/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9105 - val_loss: 211.3558
Epoch 653/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8250 - val_loss: 211.6948
Epoch 654/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9065 - val_loss: 211.0264
Epoch 655/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7382 - val_loss: 210.8220
Epoch 656/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7979 - val_loss: 211.2385
Epoch 657/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8108 - val_loss: 211.8849
Epoch 658/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9418 - val_loss: 211.5854
Epoch 659/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9114 - val_loss: 211.3121
Epoch 660/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8344 - val_loss: 211.5313
Epoch 661/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9023 - val_loss: 211.0344
Epoch 662/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8020 - val_loss: 210.7979
Epoch 663/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7673 - val_loss: 211.6118
Epoch 664/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9487 - val_loss: 211.4686
Epoch 665/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8685 - val_loss: 211.1255
Epoch 666/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8396 - val_loss: 209.7402
Epoch 667/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0044 - val_loss: 209.9845
Epoch 668/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8039 - val_loss: 210.7123
Epoch 669/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9008 - val_loss: 210.9141
Epoch 670/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8114 - val_loss: 211.4373
Epoch 671/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8873 - val_loss: 210.2924
Epoch 672/1000
12/12 [==============================] - 0s 7ms/step - loss: 177.7560 - val_loss: 210.3748
Epoch 673/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8064 - val_loss: 210.1320
Epoch 674/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7967 - val_loss: 210.8765
Epoch 675/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7446 - val_loss: 211.7457
Epoch 676/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9684 - val_loss: 211.2861
Epoch 677/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8457 - val_loss: 211.1554
Epoch 678/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7906 - val_loss: 211.9424
Epoch 679/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9975 - val_loss: 212.2414
Epoch 680/1000
```

```
12/12 [==============================] - 0s 4ms/step - loss: 178.0047 - val_loss: 212.1269
Epoch 681/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9777 - val_loss: 211.3270
Epoch 682/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9028 - val_loss: 211.2835
Epoch 683/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0009 - val_loss: 210.4143
Epoch 684/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7383 - val_loss: 210.1091
Epoch 685/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8113 - val_loss: 210.2552
Epoch 686/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7857 - val_loss: 210.3852
Epoch 687/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7063 - val_loss: 210.3969
Epoch 688/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7990 - val_loss: 210.9084
Epoch 689/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8537 - val_loss: 209.9666
Epoch 690/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7930 - val_loss: 210.3487
Epoch 691/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8447 - val_loss: 210.3583
Epoch 692/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7872 - val_loss: 209.7259
Epoch 693/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8843 - val_loss: 209.7527
Epoch 694/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9122 - val_loss: 210.1129
Epoch 695/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8043 - val_loss: 210.5315
Epoch 696/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8082 - val_loss: 210.3310
Epoch 697/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7733 - val_loss: 210.9920
Epoch 698/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7885 - val_loss: 210.2504
Epoch 699/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8197 - val_loss: 209.7773
Epoch 700/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8345 - val_loss: 209.7402
Epoch 701/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9382 - val_loss: 210.2216
Epoch 702/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8041 - val_loss: 210.3285
Epoch 703/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8561 - val_loss: 210.8375
Epoch 704/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7847 - val_loss: 210.4986
Epoch 705/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7820 - val_loss: 209.6934
Epoch 706/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.9919 - val_loss: 209.8819
Epoch 707/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9498 - val_loss: 209.8665
Epoch 708/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9553 - val_loss: 210.3207
Epoch 709/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7784 - val_loss: 211.2658
Epoch 710/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8323 - val_loss: 211.0717
Epoch 711/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8678 - val_loss: 210.9185
Epoch 712/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7978 - val_loss: 210.5578
```

```
Epoch 713/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8372 - val_loss: 210.4967
Epoch 714/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7979 - val_loss: 210.5566
Epoch 715/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7970 - val_loss: 210.7067
Epoch 716/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8074 - val_loss: 211.0138
Epoch 717/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8007 - val_loss: 210.4030
Epoch 718/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8080 - val_loss: 211.1002
Epoch 719/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7590 - val_loss: 211.3223
Epoch 720/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8573 - val_loss: 211.4678
Epoch 721/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8760 - val_loss: 211.4671
Epoch 722/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8185 - val_loss: 210.6615
Epoch 723/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8034 - val_loss: 210.2053
Epoch 724/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8717 - val_loss: 210.8652
Epoch 725/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8503 - val_loss: 210.2247
Epoch 726/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7427 - val_loss: 210.2017
Epoch 727/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8089 - val_loss: 209.4950
Epoch 728/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9755 - val_loss: 209.5913
Epoch 729/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9730 - val_loss: 210.3041
Epoch 730/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8587 - val_loss: 209.6268
Epoch 731/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0121 - val_loss: 209.3753
Epoch 732/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1747 - val_loss: 209.2327
Epoch 733/1000
12/12 [==============================] - 0s 6ms/step - loss: 178.3418 - val_loss: 209.8168
Epoch 734/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9019 - val_loss: 209.9978
Epoch 735/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7971 - val_loss: 210.0244
Epoch 736/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8517 - val_loss: 210.1284
Epoch 737/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7810 - val_loss: 210.0037
Epoch 738/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9201 - val_loss: 210.1789
Epoch 739/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7547 - val_loss: 210.4330
Epoch 740/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8231 - val_loss: 210.8288
Epoch 741/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8158 - val_loss: 210.5246
Epoch 742/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7860 - val_loss: 210.6842
Epoch 743/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8422 - val_loss: 210.1824
Epoch 744/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7995 - val_loss: 210.3204
Epoch 745/1000
```

```
12/12 [==============================] - 0s 5ms/step - loss: 177.7983 - val_loss: 210.0786
Epoch 746/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8580 - val_loss: 210.5917
Epoch 747/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7550 - val_loss: 211.1794
Epoch 748/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7668 - val_loss: 209.9713
Epoch 749/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9134 - val_loss: 210.5277
Epoch 750/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7914 - val_loss: 210.1779
Epoch 751/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8606 - val_loss: 209.9923
Epoch 752/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7827 - val_loss: 210.3349
Epoch 753/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7875 - val_loss: 210.1750
Epoch 754/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7666 - val_loss: 210.3422
Epoch 755/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7671 - val_loss: 209.8844
Epoch 756/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9126 - val_loss: 209.8849
Epoch 757/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8418 - val_loss: 210.0795
Epoch 758/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8509 - val_loss: 210.5423
Epoch 759/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7522 - val_loss: 210.4291
Epoch 760/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7614 - val_loss: 210.2076
Epoch 761/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7449 - val_loss: 210.4316
Epoch 762/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7205 - val_loss: 210.2415
Epoch 763/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7943 - val_loss: 209.7750
Epoch 764/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8413 - val_loss: 210.8128
Epoch 765/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8009 - val_loss: 210.3057
Epoch 766/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7502 - val_loss: 210.9986
Epoch 767/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7638 - val_loss: 210.4581
Epoch 768/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8329 - val_loss: 210.0141
Epoch 769/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8201 - val_loss: 209.8542
Epoch 770/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9704 - val_loss: 209.9045
Epoch 771/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8562 - val_loss: 209.5649
Epoch 772/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9639 - val_loss: 209.9068
Epoch 773/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8363 - val_loss: 209.9001
Epoch 774/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9618 - val_loss: 210.1623
Epoch 775/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7418 - val_loss: 210.3452
Epoch 776/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7525 - val_loss: 210.6155
Epoch 777/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7722 - val_loss: 211.2848
```

```
Epoch 778/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8300 - val_loss: 210.2093
Epoch 779/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8266 - val_loss: 209.9088
Epoch 780/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8339 - val_loss: 210.4144
Epoch 781/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7173 - val_loss: 210.4138
Epoch 782/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7494 - val_loss: 210.5578
Epoch 783/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8571 - val_loss: 209.9826
Epoch 784/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9250 - val_loss: 210.2830
Epoch 785/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7513 - val_loss: 210.9780
Epoch 786/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8387 - val_loss: 209.8767
Epoch 787/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8533 - val_loss: 210.1796
Epoch 788/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8337 - val_loss: 212.1923
Epoch 789/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0321 - val_loss: 211.6652
Epoch 790/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9027 - val_loss: 211.9838
Epoch 791/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.9562 - val_loss: 212.2229
Epoch 792/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0129 - val_loss: 211.3742
Epoch 793/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9034 - val_loss: 211.8605
Epoch 794/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9392 - val_loss: 212.0019
Epoch 795/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.9632 - val_loss: 211.8827
Epoch 796/1000
12/12 [==============================] - 0s 7ms/step - loss: 177.9142 - val_loss: 211.2680
Epoch 797/1000
12/12 [==============================] - 0s 7ms/step - loss: 177.7872 - val_loss: 210.8917
Epoch 798/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7650 - val_loss: 210.7331
Epoch 799/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7994 - val_loss: 210.2408
Epoch 800/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8764 - val_loss: 209.4823
Epoch 801/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0817 - val_loss: 209.6853
Epoch 802/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9226 - val_loss: 209.8990
Epoch 803/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8651 - val_loss: 210.5508
Epoch 804/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7872 - val_loss: 211.1495
Epoch 805/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9176 - val_loss: 211.6098
Epoch 806/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8395 - val_loss: 212.0258
Epoch 807/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9449 - val_loss: 212.4771
Epoch 808/1000
12/12 [==============================] - 0s 6ms/step - loss: 178.0854 - val_loss: 210.5402
Epoch 809/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8126 - val_loss: 210.4999
Epoch 810/1000
```

```
12/12 [==============================] - 0s 4ms/step - loss: 177.7720 - val_loss: 210.7351
Epoch 811/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7578 - val_loss: 210.6323
Epoch 812/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7749 - val_loss: 210.4179
Epoch 813/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7852 - val_loss: 210.5145
Epoch 814/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8386 - val_loss: 210.7907
Epoch 815/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8524 - val_loss: 210.8671
Epoch 816/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7569 - val_loss: 211.3586
Epoch 817/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7810 - val_loss: 210.4730
Epoch 818/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7433 - val_loss: 211.4780
Epoch 819/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8998 - val_loss: 211.2317
Epoch 820/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8383 - val_loss: 211.2451
Epoch 821/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8712 - val_loss: 211.1170
Epoch 822/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7870 - val_loss: 210.1088
Epoch 823/1000
12/12 [==============================] - 0s 7ms/step - loss: 177.7707 - val_loss: 209.9405
Epoch 824/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8744 - val_loss: 210.7598
Epoch 825/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8969 - val_loss: 211.5643
Epoch 826/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8377 - val_loss: 211.8475
Epoch 827/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9796 - val_loss: 211.8076
Epoch 828/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8717 - val_loss: 211.4148
Epoch 829/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7967 - val_loss: 210.9016
Epoch 830/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7619 - val_loss: 210.7204
Epoch 831/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8162 - val_loss: 211.1766
Epoch 832/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8022 - val_loss: 210.7979
Epoch 833/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8140 - val_loss: 211.5996
Epoch 834/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9136 - val_loss: 211.7486
Epoch 835/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8283 - val_loss: 211.5604
Epoch 836/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8667 - val_loss: 210.6348
Epoch 837/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7740 - val_loss: 211.5822
Epoch 838/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9474 - val_loss: 211.2820
Epoch 839/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8684 - val_loss: 210.5175
Epoch 840/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8516 - val_loss: 210.6775
Epoch 841/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7234 - val_loss: 210.1542
Epoch 842/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.8348 - val_loss: 209.5181
```

```
Epoch 843/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9786 - val_loss: 209.5968
Epoch 844/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9672 - val_loss: 209.5098
Epoch 845/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0768 - val_loss: 209.7897
Epoch 846/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0067 - val_loss: 209.9644
Epoch 847/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8708 - val_loss: 209.5118
Epoch 848/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0220 - val_loss: 209.9844
Epoch 849/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8236 - val_loss: 209.5748
Epoch 850/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9807 - val_loss: 210.4956
Epoch 851/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7704 - val_loss: 210.6828
Epoch 852/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8279 - val_loss: 210.2574
Epoch 853/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8371 - val_loss: 211.1832
Epoch 854/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8333 - val_loss: 211.5014
Epoch 855/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7919 - val_loss: 211.3672
Epoch 856/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8100 - val_loss: 211.2871
Epoch 857/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7965 - val_loss: 211.4827
Epoch 858/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8278 - val_loss: 210.7350
Epoch 859/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7203 - val_loss: 210.4204
Epoch 860/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7323 - val_loss: 209.9638
Epoch 861/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8017 - val_loss: 209.8088
Epoch 862/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9539 - val_loss: 210.1588
Epoch 863/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8170 - val_loss: 210.6621
Epoch 864/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7481 - val_loss: 210.6761
Epoch 865/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8403 - val_loss: 210.9573
Epoch 866/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7343 - val_loss: 210.5724
Epoch 867/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8456 - val_loss: 210.9209
Epoch 868/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7850 - val_loss: 210.1255
Epoch 869/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8121 - val_loss: 209.4220
Epoch 870/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0768 - val_loss: 209.3065
Epoch 871/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.2404 - val_loss: 209.7757
Epoch 872/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9064 - val_loss: 211.0721
Epoch 873/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8207 - val_loss: 209.5540
Epoch 874/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0315 - val_loss: 209.3402
Epoch 875/1000
```

```
12/12 [==============================] - 0s 5ms/step - loss: 178.2944 - val_loss: 209.4651
Epoch 876/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0710 - val_loss: 209.7879
Epoch 877/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8909 - val_loss: 210.4778
Epoch 878/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8273 - val_loss: 210.4830
Epoch 879/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8372 - val_loss: 210.4935
Epoch 880/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7862 - val_loss: 209.8593
Epoch 881/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8343 - val_loss: 209.9221
Epoch 882/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8853 - val_loss: 209.5956
Epoch 883/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9609 - val_loss: 209.4363
Epoch 884/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1089 - val_loss: 209.4935
Epoch 885/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0770 - val_loss: 209.7229
Epoch 886/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9027 - val_loss: 209.5957
Epoch 887/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9898 - val_loss: 210.0305
Epoch 888/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8124 - val_loss: 211.1195
Epoch 889/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7950 - val_loss: 210.2550
Epoch 890/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7811 - val_loss: 211.1083
Epoch 891/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7899 - val_loss: 210.5536
Epoch 892/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7648 - val_loss: 210.2769
Epoch 893/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7906 - val_loss: 209.9429
Epoch 894/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8092 - val_loss: 210.7462
Epoch 895/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8129 - val_loss: 210.2311
Epoch 896/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7740 - val_loss: 209.7401
Epoch 897/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8816 - val_loss: 209.8147
Epoch 898/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8248 - val_loss: 209.9553
Epoch 899/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9514 - val_loss: 210.9214
Epoch 900/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7800 - val_loss: 211.1915
Epoch 901/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8261 - val_loss: 210.6730
Epoch 902/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7832 - val_loss: 210.9025
Epoch 903/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7887 - val_loss: 210.0867
Epoch 904/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8060 - val_loss: 210.1806
Epoch 905/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8246 - val_loss: 210.3770
Epoch 906/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7666 - val_loss: 209.7815
Epoch 907/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0409 - val_loss: 209.7493
```

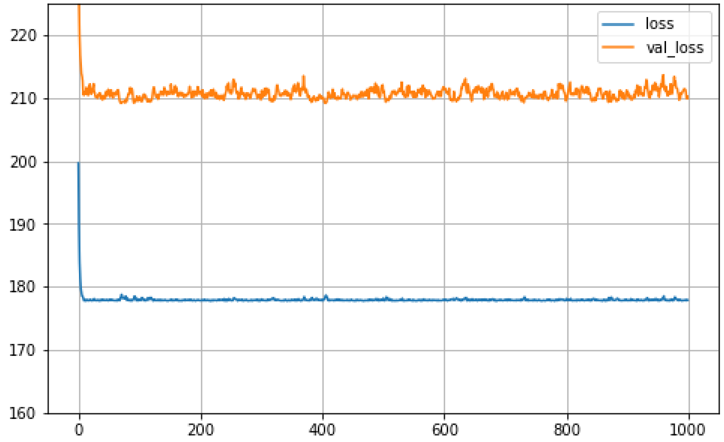
```
Epoch 908/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8989 - val_loss: 209.8042
Epoch 909/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8298 - val_loss: 209.6918
Epoch 910/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9205 - val_loss: 209.9283
Epoch 911/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8728 - val_loss: 210.3483
Epoch 912/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8094 - val_loss: 210.4755
Epoch 913/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8189 - val_loss: 209.9874
Epoch 914/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7954 - val_loss: 211.0811
Epoch 915/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8270 - val_loss: 210.8424
Epoch 916/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7171 - val_loss: 210.7344
Epoch 917/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8363 - val_loss: 211.9761
Epoch 918/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0016 - val_loss: 211.3578
Epoch 919/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8090 - val_loss: 210.3247
Epoch 920/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8114 - val_loss: 210.9027
Epoch 921/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7676 - val_loss: 211.4384
Epoch 922/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8151 - val_loss: 211.7466
Epoch 923/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8960 - val_loss: 210.7992
Epoch 924/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8445 - val_loss: 210.5475
Epoch 925/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7684 - val_loss: 210.4469
Epoch 926/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8212 - val_loss: 210.6506
Epoch 927/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7633 - val_loss: 210.2980
Epoch 928/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7496 - val_loss: 210.4197
Epoch 929/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8591 - val_loss: 211.2950
Epoch 930/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9384 - val_loss: 211.7376
Epoch 931/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8710 - val_loss: 211.6176
Epoch 932/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9019 - val_loss: 212.8441
Epoch 933/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.1801 - val_loss: 212.7463
Epoch 934/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1869 - val_loss: 211.6092
Epoch 935/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8705 - val_loss: 210.8233
Epoch 936/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7676 - val_loss: 210.2130
Epoch 937/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7547 - val_loss: 210.0649
Epoch 938/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9220 - val_loss: 209.9533
Epoch 939/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8605 - val_loss: 211.0061
Epoch 940/1000
```

```
12/12 [==============================] - 0s 5ms/step - loss: 177.9059 - val_loss: 210.5234
Epoch 941/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8229 - val_loss: 210.6394
Epoch 942/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7994 - val_loss: 211.9093
Epoch 943/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9604 - val_loss: 211.5109
Epoch 944/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8250 - val_loss: 211.3045
Epoch 945/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8420 - val_loss: 211.1781
Epoch 946/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8945 - val_loss: 212.3584
Epoch 947/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0687 - val_loss: 211.8467
Epoch 948/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9187 - val_loss: 212.2213
Epoch 949/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0198 - val_loss: 211.2469
Epoch 950/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9485 - val_loss: 211.0766
Epoch 951/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7566 - val_loss: 211.5999
Epoch 952/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8609 - val_loss: 211.0358
Epoch 953/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7491 - val_loss: 210.1308
Epoch 954/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8401 - val_loss: 210.8650
Epoch 955/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7902 - val_loss: 211.7298
Epoch 956/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8624 - val_loss: 212.1658
Epoch 957/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0108 - val_loss: 211.6908
Epoch 958/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.9378 - val_loss: 212.5523
Epoch 959/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.1252 - val_loss: 213.2943
Epoch 960/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.3083 - val_loss: 213.6196
Epoch 961/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.4932 - val_loss: 211.7335
Epoch 962/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8701 - val_loss: 211.5059
Epoch 963/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8986 - val_loss: 211.1223
Epoch 964/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8354 - val_loss: 211.8216
Epoch 965/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9066 - val_loss: 210.8818
Epoch 966/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7852 - val_loss: 210.9052
Epoch 967/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7407 - val_loss: 210.3287
Epoch 968/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7863 - val_loss: 211.0233
Epoch 969/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8588 - val_loss: 210.4804
Epoch 970/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7444 - val_loss: 210.7686
Epoch 971/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7628 - val_loss: 211.0087
Epoch 972/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8608 - val_loss: 212.1816
```

```
Epoch 973/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0102 - val_loss: 211.4408
Epoch 974/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8234 - val_loss: 210.5977
Epoch 975/1000
12/12 [==============================] - 0s 6ms/step - loss: 177.7744 - val_loss: 211.0320
Epoch 976/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7652 - val_loss: 211.9348
Epoch 977/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0080 - val_loss: 212.1933
Epoch 978/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9700 - val_loss: 213.3583
Epoch 979/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.3427 - val_loss: 212.1448
Epoch 980/1000
12/12 [==============================] - 0s 4ms/step - loss: 178.0760 - val_loss: 212.5755
Epoch 981/1000
12/12 [==============================] - 0s 5ms/step - loss: 178.0918 - val_loss: 211.6318
Epoch 982/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.9109 - val_loss: 211.5326
Epoch 983/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8684 - val_loss: 211.4358
Epoch 984/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9015 - val_loss: 210.8017
Epoch 985/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7809 - val_loss: 210.5305
Epoch 986/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8029 - val_loss: 209.6333
Epoch 987/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9496 - val_loss: 209.7881
Epoch 988/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8941 - val_loss: 209.9556
Epoch 989/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8504 - val_loss: 210.7908
Epoch 990/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7418 - val_loss: 210.6284
Epoch 991/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7523 - val_loss: 210.7472
Epoch 992/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.7618 - val_loss: 211.0093
Epoch 993/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8304 - val_loss: 211.1824
Epoch 994/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8217 - val_loss: 211.3862
Epoch 995/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8011 - val_loss: 211.2732
Epoch 996/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.8086 - val_loss: 211.3299
Epoch 997/1000
12/12 [==============================] - 0s 4ms/step - loss: 177.9109 - val_loss: 210.9468
Epoch 998/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.7740 - val_loss: 209.9393
Epoch 999/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8205 - val_loss: 210.2593
Epoch 1000/1000
12/12 [==============================] - 0s 5ms/step - loss: 177.8396 - val_loss: 209.8985
```

In [80]:

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.gca().set_ylim(160, 225)
plt.grid(True)
plt.show()
```



Compute Generalization Error

GradientBoostingRegressor Model MSE

In [81]:

```
y_pred_gbrt = gbrt.predict(X_test)
gbrt_mse = mean_squared_error(y_test, y_pred_gbrt)
print("GradientBoostingRegressor Model MSE:", round(gbrt_mse,3))
```

```
GradientBoostingRegressor Model MSE: 121.497
```

GradientBoostingRegressor Scaled Model MSE

In [82]:

```
y_pred_gbrt_scaled = gbrt_scaled.predict(X_test_scaled)
gbrt_scaled_mse = mean_squared_error(y_test, y_pred_gbrt_scaled)
print("GradientBoostingRegressor Scaled Model MSE:", round(gbrt_scaled_mse,3))
```

```
GradientBoostingRegressor Scaled Model MSE: 121.703
```

GradientBoostingRegressor PCA Model MSE

In [83]:

```
y_pred_gbrt_PCA = gbrt.predict(X_test)
gbrt_PCA_mse = mean_squared_error(y_test, y_pred_gbrt_PCA)
print("GradientBoostingRegressor PCA Model MSE:", round(gbrt_PCA_mse,3))
```

```
GradientBoostingRegressor PCA Model MSE: 121.497
```

GradientBoostingRegressor PCA Scaled Model MSE

In [84]:

```
y_pred_gbrt_PCA_scaled = gbrt_scaled.predict(X_test_scaled)
gbrt_PCA_scaled_mse = mean_squared_error(y_test, y_pred_gbrt_PCA_scaled)
print("GradientBoostingRegressor PCA Scaled Model MSE:", round(gbrt_PCA_scaled_mse,3))
```

```
GradientBoostingRegressor PCA Scaled Model MSE: 121.703
```

RandomForestRegressor Model MSE

In [85]:
```
y_pred_rnd = rnd_reg.predict(X_test)
rnd_mse = mean_squared_error(y_test, y_pred_rnd)
print("RandomForestRegressor Model MSE:", round(rnd_mse,3))
```

```
RandomForestRegressor Model MSE: 122.449
```

RandomForestRegressor Scaled Model MSE

In [86]:
```
y_pred_rnd_scaled = rnd_reg_scaled.predict(X_test_scaled)
rnd_scaled_mse = mean_squared_error(y_test, y_pred_rnd_scaled)
print("RandomForestRegressor Scaled Model MSE:", round(rnd_scaled_mse,3))
```

```
RandomForestRegressor Scaled Model MSE: 122.489
```

RandomForestRegressor PCA Model MSE

In [87]:
```
y_pred_rnd_PCA = rnd_reg.predict(X_test)
rnd_PCA_mse = mean_squared_error(y_test, y_pred_rnd_PCA)
print("RandomForestRegressor PCA Model MSE:", round(rnd_PCA_mse,3))
```

```
RandomForestRegressor PCA Model MSE: 122.449
```

RandomForestRegressor PCA Scaled Model MSE

In [88]:
```
y_pred_rnd_PCA_scaled = rnd_reg_scaled.predict(X_test_scaled)
rnd_scaled_PCA_mse = mean_squared_error(y_test, y_pred_rnd_PCA_scaled)
print("RandomForestRegressor PCA Scaled Model MSE:", round(rnd_scaled_PCA_mse,3))
```

```
RandomForestRegressor PCA Scaled Model MSE: 122.489
```

DecisionTreeRegressor Model MSE

In [89]:
```
y_pred_tree = tree_reg.predict(X_test)
tree_mse = mean_squared_error(y_test, y_pred_tree)
print("DecisionTreeRegressor Model MSE:", round(tree_mse,3))
```

```
DecisionTreeRegressor Model MSE: 124.925
```

DecisionTreeRegressor Scaled Model MSE

In [90]:
```
y_pred_tree_scaled = tree_reg_scaled.predict(X_test_scaled)
tree_scaled_mse = mean_squared_error(y_test, y_pred_tree_scaled)
print("DecisionTreeRegressor Scaled Model MSE:", round(tree_scaled_mse,3))
```

```
DecisionTreeRegressor Scaled Model MSE: 124.925
```

DecisionTreeRegressor PCA Model MSE

In [91]:
```
y_pred_tree_PCA = tree_reg.predict(X_test)
tree_PCA_mse = mean_squared_error(y_test, y_pred_tree_PCA)
print("DecisionTreeRegressor Model MSE:", round(tree_PCA_mse,3))
```

```
DecisionTreeRegressor Model MSE: 124.925
```

DecisionTreeRegressor PCA Scaled Model MSE

In [92]:
```
y_pred_tree_PCA_scaled = tree_reg_scaled.predict(X_test_scaled)
tree_scaled_PCA_mse = mean_squared_error(y_test, y_pred_tree_PCA_scaled)
print("DecisionTreeRegressor Scaled Model MSE:", round(tree_scaled_PCA_mse,3))
```

```
DecisionTreeRegressor Scaled Model MSE: 124.925
```

KNeighborsRegressor

In [93]:
```
y_pred_knn = knn_reg.predict(X_test)
knn_mse = mean_squared_error(y_test, y_pred_knn)
print("KNeighborsRegressor Model MSE:", round(knn_mse,3))
```

```
KNeighborsRegressor Model MSE: 123.247
```

KNeighborsRegressor w/Scaling

In [94]:
```
y_pred_knn_scaled = knn_scaled_reg.predict(X_test_scaled)
knn_scaled_mse = mean_squared_error(y_test, y_pred_knn_scaled)
print("KNeighborsRegressor Scaled Model MSE:", round(knn_scaled_mse,3))
```

```
KNeighborsRegressor Scaled Model MSE: 196.962
```

VotingRegressor

In [95]:
```
y_pred_voting_reg = voting_reg.predict(X_test_scaled)
voting_reg_mse = mean_squared_error(y_test, y_pred_voting_reg)
print("VotingRegressor Model MSE:", round(voting_reg_mse,3))
```

```
VotingRegressor Model MSE: 170.702
```

Artificial Neural Networks MSE

In [96]:
```
ANN_mse = model_ANN.evaluate(X_test,y_test)
print(ANN_mse)
```

```
5/5 [==============================] - 0s 1ms/step - loss: 166.5900
166.5900115966797
```

Compare Generalization across models

In [97]:
```
print("GradientBoostingRegressor Model MSE:", round(gbrt_mse,3))
print("GradientBoostingRegressor Scaled Model MSE:", round(gbrt_scaled_mse,3))
print("GradientBoostingRegressor PCA Model MSE:", round(gbrt_PCA_mse,3))
print("GradientBoostingRegressor PCA Scaled Model MSE:", round(gbrt_PCA_scaled_mse,3))
print("RandomForestRegressor Model MSE:", round(rnd_mse,3))
print("RandomForestRegressor Scaled Model MSE:", round(rnd_scaled_mse,3))
print("RandomForestRegressor PCA Model MSE:", round(rnd_PCA_mse,3))
print("RandomForestRegressor PCA Scaled Model MSE:", round(rnd_scaled_PCA_mse,3))
```

```
print("DecisionTreeRegressor Model MSE:", round(tree_mse,3))
print("DecisionTreeRegressor Scaled Model MSE:", round(tree_scaled_mse,3))
print("DecisionTreeRegressor Model MSE:", round(tree_PCA_mse,3))
print("DecisionTreeRegressor Scaled Model MSE:", round(tree_scaled_PCA_mse,3))
print("KNeighborsRegressor Model MSE:", round(knn_mse,3))
print("KNeighborsRegressor Scaled Model MSE:", round(knn_scaled_mse,3))
print("VotingRegressor Model MSE:", round(voting_reg_mse,3))
print("Artificial Neural Network Scaled Model MSE:",round(ANN_mse,3))
print("Better Seed Benchmark Model MSE:", round(mse_BetterSeed_Scaled,3))
print("Better Record Benchmark Model MSE:", round(mse_BetterRecord_Scaled,3))
```

```
GradientBoostingRegressor Model MSE: 121.497
GradientBoostingRegressor Scaled Model MSE: 121.703
GradientBoostingRegressor PCA Model MSE: 121.497
GradientBoostingRegressor PCA Scaled Model MSE: 121.703
RandomForestRegressor Model MSE: 122.449
RandomForestRegressor Scaled Model MSE: 122.489
RandomForestRegressor PCA Model MSE: 122.449
RandomForestRegressor PCA Scaled Model MSE: 122.489
DecisionTreeRegressor Model MSE: 124.925
DecisionTreeRegressor Scaled Model MSE: 124.925
DecisionTreeRegressor Model MSE: 124.925
DecisionTreeRegressor Scaled Model MSE: 124.925
KNeighborsRegressor Model MSE: 123.247
KNeighborsRegressor Scaled Model MSE: 196.962
VotingRegressor Model MSE: 170.702
Artificial Neural Network Scaled Model MSE: 166.59
Better Seed Benchmark Model MSE: 139.056
Better Record Benchmark Model MSE: 142.423
```

Item	Models	No Scaling	Scaling	No Scaling PCA	Scaling PCA
MSE	Gradient Boosting	**121.497**	121.703	121.497	121.703
	Random Forest	122.449	122.489	122.449	122.489
	Decision Tree	124.925	124.925	124.925	124.925
	KNeighbors	123.247	196.962		
	ANN	166.590			
	VotingReg	170.702			
	BM Better Seed	139.056			
	BM Better Record	142.423			

Model Decision

The `Gradient Boosting Model` with no scaling was the lowest MSE and is the model I have chosen. The PCA for that non-scaled model didn't seem to make a signifcant change in the feature set.

The other individual models, according to the MSE, performed very similarly using their respective optimal parameters when fitting. The Voting Regression Model didn't perform as well as I initially thought it would. Upon further reflection that makes sense to me because the individual model optimal parameters and predictions may not actually combine well with the other models to produce a more optimal prediction, with many times the differing models potentially pulling and skewing the results in multiple directions for each prediction.

I was able to show that the individual models I ran, based only on MSE, would be beneficial to use over the benchmark strategies when filling out a tournament bracket to only select the team with the better seed or record.

The features and distribution of those features didn't need to be scaled or narrowed using PCA, and so there wasn't any clear benefit reflected in the MSE using those processes. As a result, the later models I created and fitted I decided that those processes didn't need to be included as they didn't add any clear benefit.

Creating Flask Application

Wasn't too familiar with Flask so used the source provided to me when discussing my project proposal. https://medium.com/analytics-vidhya/deploying-a-machine-learning-model-using-flask-for-beginners-674944714b86

In [98]:

```
#serializing model to a file called model.pkl
#using gbrt as the chosen model
pickle.dump(gbrt,open("model.pkl","wb"))
```

In [99]:

```
#creating instance of the class
app = Flask(__name__,template_folder='templates')

#to tell flask what url should trigger the function index()
@app.route('/')
@app.route('/index')
def index():
    return flask.render_template('index.html')
```

Index and Result html files needed will be stored in templates folder that will also be stored in the Google Drive.

Finding feature list that will be the inputs on the application

In [100…

```
X_train.columns
```

Out[100…

```
Index(['assist_percentage', 'block_percentage', 'offensive_rating',
       'offensive_rebound_percentage', 'opp_assist_percentage',
       'opp_block_percentage', 'opp_free_throw_attempt_rate',
       'opp_free_throw_percentage', 'opp_free_throws_per_field_goal_attempt',
       'opp_offensive_rebound_percentage', 'opp_steal_percentage',
       'opp_three_point_field_goal_percentage',
       'opp_two_point_field_goal_percentage', 'opp_total_rebound_percentage',
       'opp_true_shooting_percentage', 'pace', 'simple_rating_system',
       'steal_percentage', 'strength_of_schedule', 'three_point_attempt_rate',
       'three_point_field_goal_percentage', 'two_point_field_goal_percentage',
       'true_shooting_percentage', 'turnover_percentage', 'win_percentage',
       'seed_difference'],
      dtype='object')
```

In [101…

```
#prediction function
def InputPredictor(to_predict_inputs):
    to_predict = np.array(to_predict_inputs).reshape(1,26)
    loaded_model = pickle.load(open("model.pkl","rb"))
    result = loaded_model.predict(to_predict)
    return result[0]

@app.route('/result',methods = ['POST'])
def result():
    if request.method == 'POST':
        to_predict_inputs = request.form.values()
        to_predict_inputs = list(map(float,to_predict_inputs))
```

```
        result = InputPredictor(to_predict_inputs)

        if float(result) > 0:
            prediction = 'Home Team will win by ' + str(math.ceil(result)) + ' point(s)'
        elif float(result) < 0:
            prediction = 'Away Team will win by ' + str(abs(math.floor(result))) + ' point(s)'
        elif float(result) == 0:
            prediction = 'There will be a tie'

        return render_template("result.html",prediction = prediction)
```

In [102…

```
if __name__ == '__main__':
    app.run(debug=False)
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Apr/2022 22:03:53] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Apr/2022 22:03:53] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [23/Apr/2022 22:05:26] "POST /result HTTP/1.1" 200 -
127.0.0.1 - - [23/Apr/2022 22:05:33] "POST /result HTTP/1.1" 200 -
127.0.0.1 - - [23/Apr/2022 22:41:03] "POST /result HTTP/1.1" 200 -
```

Creating predictions using real games through the Flask Application

In [81]:

```
#Combining test set data to make predictions off of and compare to actual results
Flask_predictions = pd.concat([game_info_test, X_test,y_test], axis=1)
Flask_predictions = Flask_predictions.rename(columns={0:"Final_Score_Difference"})
Flask_predictions
```

Out[81]:

	Year	Home_Team	Away_Team	assist_percentage	block_percentage	offensive_rating	offensive_rebound_percentage	opp_assist_percentage	opp_block_percentage	opp_free_throw_attempt_rate
497	2017	Villanova	Mount St. Mary's	0.063	-0.006	18.5	0.070	0.131	-0.013	-0.105
244	2013	Syracuse	California	-0.004	0.079	6.6	0.063	0.165	0.008	0.029
552	2018	Texas A&M	Michigan	0.015	0.072	-4.9	0.085	0.139	-0.002	-0.022
213	2013	Louisville	Colorado State	0.052	0.057	-2.3	-0.030	0.059	0.001	-0.015
549	2018	TCU	Syracuse	0.153	-0.080	12.7	-0.005	-0.166	0.023	-0.009
...	...	...	...	...	...	...	...	...	...	...
287	2014	Michigan State	Delaware	0.219	0.049	1.6	0.031	0.054	-0.016	0.023
369	2015	West Virginia	Buffalo	0.050	-0.026	-3.6	0.057	0.040	0.022	0.212
79	2011	Florida	UCLA	-0.056	-0.041	6.5	0.012	-0.015	0.006	-0.052
23	2010	Marquette	Washington	0.086	-0.047	3.4	-0.057	0.101	0.037	-0.152

	Year	Home_Team	Away_Team	assist_percentage	block_percentage	offensive_rating	offensive_rebound_percentage	opp_assist_percentage	opp_block_percentage	opp_free_throw_attempt_rate
583	2019	Iowa State	Ohio State	-0.048	0.055	9.5	0.008	-0.002	-0.018	-0.082

158 rows × 30 columns

First Game Prediction

In [82]:

```
#Getting first game shown in the test set
Flask_predictions.iloc[0]
```

Out[82]:

```
Year                                              2017
Home_Team                                    Villanova
Away_Team                             Mount St. Mary's
assist_percentage                                0.063
block_percentage                                -0.006
offensive_rating                                  18.5
offensive_rebound_percentage                      0.07
opp_assist_percentage                            0.131
opp_block_percentage                            -0.013
opp_free_throw_attempt_rate                     -0.105
opp_free_throw_percentage                        0.043
opp_free_throws_per_field_goal_attempt           -0.06
opp_offensive_rebound_percentage                -0.072
opp_steal_percentage                            -0.015
opp_three_point_field_goal_percentage            -0.03
opp_two_point_field_goal_percentage             -0.004
opp_total_rebound_percentage                    -0.085
opp_true_shooting_percentage                    -0.019
pace                                              -2.8
simple_rating_system                             28.09
steal_percentage                                  0.02
strength_of_schedule                             12.79
three_point_attempt_rate                         0.064
three_point_field_goal_percentage                0.011
two_point_field_goal_percentage                  0.098
true_shooting_percentage                         0.076
turnover_percentage                             -0.021
win_percentage                                   0.333
seed_difference                                    -15
Final_Score_Difference                              20
Name: 497, dtype: object
```

NCAA Tournament Game Prediction Based on Home - Away Season Stat Difference

Difference in Assist Percentage .063
Difference in Block Percentage -.006
Difference in Offensive Rating 18.5
Difference in Offensive Rebound Percentage .07
Difference in Opponent Assist Percentage .131
Difference in Opponent Block Percentage -.013
Difference in Opponent Free Throw Attempt Rate -.105
Difference in Opponent Free Throw Percentage .043
Difference in Opponent Free Throw Per Field Goal Attempt -.06
Difference in Opponent Offensive Rebound Percentage -.072
Difference in Opponent Steal Percentage -.015
Difference in Opponent Three Point Field Goal Percentage -.03
Difference in Opponent Two Point Field Goal Percentage -.004
Difference in Opponent Total Rebound Percentage -.085
Difference in Opponent True Shooting Percentage -.019
Difference in Pace -2.8
Difference in Simple Rating System 28.09
Difference in Steal Percentage .02
Difference in Strength of Schedule 12.79
Difference in Three Point Attempt Rate .064
Difference in Three Point Field Goal Percentage .011
Difference in Two Point Field Goal Percentage .098
Difference in True Shooting Percentage .076
Difference in Turnover Percentage -.021
Difference in Win Percentage .333
Difference in Seed -15
Submit

Inputting above data into prediction tool:

Home Team will win by 18 point(s)

Prediction through Flask: In line with the 20 point actual win by Villanova(Home Team)

Confirming Flask model is working as intended

In [92]:
```
X_test.iloc[0]
```

Out[92]:
```
assist_percentage                        0.063
block_percentage                        -0.006
offensive_rating                        18.500
offensive_rebound_percentage             0.070
opp_assist_percentage                    0.131
opp_block_percentage                    -0.013
opp_free_throw_attempt_rate             -0.105
opp_free_throw_percentage                0.043
opp_free_throws_per_field_goal_attempt  -0.060
opp_offensive_rebound_percentage        -0.072
opp_steal_percentage                    -0.015
opp_three_point_field_goal_percentage   -0.030
```

```
opp_two_point_field_goal_percentage       -0.004
opp_total_rebound_percentage              -0.085
opp_true_shooting_percentage              -0.019
pace                                      -2.800
simple_rating_system                      28.090
steal_percentage                           0.020
strength_of_schedule                      12.790
three_point_attempt_rate                   0.064
three_point_field_goal_percentage          0.011
two_point_field_goal_percentage            0.098
true_shooting_percentage                   0.076
turnover_percentage                       -0.021
win_percentage                             0.333
seed_difference                          -15.000
Name: 497, dtype: float64
```

In [93]:

```
# This result is rounding up to the win by 18 shown above through the application.
y_pred_gbrt[0]
```

Out[93]:

```
17.553166233183706
```

Second Game Prediction

In [83]:

```
Flask_predictions.iloc[1]
```

Out[83]:

```
Year                                            2013
Home_Team                                   Syracuse
Away_Team                                 California
assist_percentage                             -0.004
block_percentage                               0.079
offensive_rating                                 6.6
offensive_rebound_percentage                   0.063
opp_assist_percentage                          0.165
opp_block_percentage                           0.008
opp_free_throw_attempt_rate                    0.029
opp_free_throw_percentage                     -0.015
opp_free_throws_per_field_goal_attempt         0.015
opp_offensive_rebound_percentage               0.052
opp_steal_percentage                           0.017
opp_three_point_field_goal_percentage         -0.053
opp_two_point_field_goal_percentage             0.01
opp_total_rebound_percentage                   0.003
opp_true_shooting_percentage                  -0.013
pace                                            -1.3
simple_rating_system                            9.71
steal_percentage                                0.05
strength_of_schedule                            0.86
three_point_attempt_rate                       0.068
three_point_field_goal_percentage              0.033
two_point_field_goal_percentage               -0.003
true_shooting_percentage                       0.008
turnover_percentage                           -0.007
win_percentage                                 0.114
seed_difference                                   -8
Final_Score_Difference                             6
Name: 244, dtype: object
```

NCAA Tournament Game Prediction Based on Home - Away Season Stat Difference

Difference in Assist Percentage -.004
Difference in Block Percentage .079
Difference in Offensive Rating 6.6
Difference in Offensive Rebound Percentage .063
Difference in Opponent Assist Percentage .165
Difference in Opponent Block Percentage .008
Difference in Opponent Free Throw Attempt Rate .029
Difference in Opponent Free Throw Percentage -.015
Difference in Opponent Free Throw Per Field Goal Attempt .015
Difference in Opponent Offensive Rebound Percentage .052
Difference in Opponent Steal Percentage .017
Difference in Opponent Three Point Field Goal Percentage -.053
Difference in Opponent Two Point Field Goal Percentage .01
Difference in Opponent Total Rebound Percentage .003
Difference in Opponent True Shooting Percentage -.013
Difference in Pace -1.3
Difference in Simple Rating System 9.71
Difference in Steal Percentage .05
Difference in Strength of Schedule .86
Difference in Three Point Attempt Rate .068
Difference in Three Point Field Goal Percentage .033
Difference in Two Point Field Goal Percentage -.003
Difference in True Shooting Percentage .008
Difference in Turnover Percentage -.007
Difference in Win Percentage .114
Difference in Seed -8
Submit

Inputting above data into prediction tool:

Home Team will win by 8 point(s)

Prediction through Flask: In line with the 6 point actual win by Syracuse(Home Team)

Confirming Flask model is working as intended

In [94]:
```
X_test.iloc[1]
```

Out[94]:
```
assist_percentage                        -0.004
block_percentage                          0.079
offensive_rating                          6.600
offensive_rebound_percentage              0.063
opp_assist_percentage                     0.165
opp_block_percentage                      0.008
opp_free_throw_attempt_rate               0.029
opp_free_throw_percentage                -0.015
opp_free_throws_per_field_goal_attempt    0.015
opp_offensive_rebound_percentage          0.052
opp_steal_percentage                      0.017
```

```
opp_three_point_field_goal_percentage    -0.053
opp_two_point_field_goal_percentage       0.010
opp_total_rebound_percentage              0.003
opp_true_shooting_percentage             -0.013
pace                                     -1.300
simple_rating_system                      9.710
steal_percentage                          0.050
strength_of_schedule                      0.860
three_point_attempt_rate                  0.068
three_point_field_goal_percentage         0.033
two_point_field_goal_percentage          -0.003
true_shooting_percentage                  0.008
turnover_percentage                      -0.007
win_percentage                            0.114
seed_difference                          -8.000
Name: 244, dtype: float64
```

In [95]:

```
y_pred_gbrt[1]
```

Out[95]:

```
7.608803021730753
```

DTSC 691
Machine Learning
Project Proposal
Garrett Fanning

Goals of the project

The National Collegiate Basketball Association (NCAA) has over 350 colleges all competing to earn a spot in the NCAA tournament, with the ultimate goal of winning the tournament. Only 68 colleges are able to play well enough to earn a spot in the tournament. The lowest 8 teams play against each other to whittle down the field to 64 teams, from which a typical bracket style tournament can be created.

The tournament begins in March with games being played nearly on a daily basis through the entire month. The amount of games, exciting finishes, and surprising outcomes/upsets has led this whole experience to be termed as "March Madness". Fans join in on this experience by trying to guess/predict how the entire tournament will play out. It's popular for people to create groups with their family, friends, and coworkers to see who was able to be the most accurate with their picks.

For the most part, picking the winner of each game comes down to guessing or using your general basketball knowledge where it's impossible to consistently predict games correctly. The NCAA tournament at first glance may appear to have a lot of randomness, which has led many people to leverage the vast amount of college basketball data out there to create algorithms that can have more predictability than the simple guessing or just picking the consensus favorite for each game.

Kaggle has many competitions each year to give those with a data science background the opportunity to test their skills trying to create a model to accurately predict any possible matchup between teams in the tournament. There's plenty of other sponsored opportunities across the Internet for the general public to participate, such as when Warren Buffett offered $1 billion to anyone able to get a perfect bracket, which is virtually impossible. For more context here is a link explaining that opportunity:
<https://bleacherreport.com/articles/1931210-warren-buffet-will-pay-1-billion-to-fan-with-perfect-march-madness-bracket>

My goal is to build a model with better predictability than if I were to simply pick the favorite in each matchup(lower seed, #1 seed is better than #2). Once I am able to train and test a model, I hope to be able to use that model to help me with the current tournament that is about to begin.

I will use regular season data to help predict how that translates to success in the NCAA tournament. All of the features I will be using are averages or on a per game basis because all teams don't play an equal amount of games in the regular season. Many of these features will not be used if they do not appear significant or are too similar to other features. I will predict winners of matchups in the NCAA tournament through predicting a final score differential, which can be interpreted as a positive differential means one team wins or negative would mean the other team.

If I have time then I could build a visualization of a bracket that would automatically populate based on my predictions.

Data description

I'm planning on pulling from 2 sources:

1) pip install pandas sklearn sportsreference → in terminal
   from sportsreference.ncaab.teams import Teams

The above code is showing that within sklearn there is an api that I can use to pull college basketball data from a vast sports data repository.

The data I would get from here right now is around 40 features of regular season statistics with each team that participated in the NCAA tournament from 2010-2019. Ex: average points scored per game, average points allowed, rebounds per game, etc.

Features pulled from data source(All features or used for merging):
'year','name','abbreviation', 'assist_percentage', 'block_percentage', 'effective_field_goal_percentage', 'field_goal_percentage', 'free_throw_attempt_rate', 'free_throw_percentage', 'free_throws_per_field_goal_attempt', 'offensive_rating', 'offensive_rebound_percentage', 'opp_assist_percentage', 'opp_block_percentage', 'opp_effective_field_goal_percentage','opp_field_goal_percentage', 'opp_free_throw_attempt_rate', 'opp_free_throw_percentage', 'opp_free_throws_per_field_goal_attempt', 'opp_offensive_rating', 'opp_offensive_rebound_percentage','opp_steal_percentage', 'opp_three_point_attempt_rate','opp_three_point_field_goal_percentage', 'opp_two_point_field_goal_percentage','opp_total_rebound_percentage', 'opp_true_shooting_percentage', 'opp_turnover_percentage', 'pace', 'simple_rating_system', 'steal_percentage', 'strength_of_schedule', 'three_point_attempt_rate', 'three_point_field_goal_percentage', 'two_point_field_goal_percentage', 'two_point_field_goals',

'total_rebound_percentage', 'true_shooting_percentage', 'turnover_percentage','win_percentage'

2) The second source is a csv with the results of each game from each NCAA tournament dating back to 1985(Right now using only data from 2010-2019). I would use the results from these games (score differential) as the response variable to match up with the features from the prior source.
   The csv was found on:
   <https://data.world/michaelaroy/ncaa-tournament-results/workspace/file?filename=Big_Dance_CSV.csv>
   You can either make an account on the site to download the csv or access the copy I placed in the shared Google Drive folder.

Data pulled from Source:
'FinalScore_Difference' (Response), 'Seed' (Feature)
'Year' and 'Team' (Merging)

So in total roughly 10 years of data x 68 teams x 40 features would mean around 27200 observations. I'm early in the process so these numbers might slightly change with more/less years or features.

Software

I'm planning on working in Jupyter notebooks using the necessary data science or machine learning python packages. I won't need any external database software because in my notebook I'll be pulling in my data directly from the sources. I'll mainly use packages from sklearn for training and testing various models/methods. If I need to better visualize, understand, or explain results then I will use Tableau. I will use Google Drive for sharing any relevant files or code.

Analysis plan & Model Specifications

Analysis description

I've outlined my steps below that I will accomplish each week throughout the entirety of this project.

Week 1 goals

1) Research topic ideas and submit project proposal
2) Find data sources that would make interesting project topics feasible

Week 2 goals

1) Successfully import data into Jupyter Notebook
2) Merge and clean data

Much of the work this week will be trying to merge the 2 sources using the year and team. Looking at the data sources, it doesn't appear there will be any issues in cleaning the data, with the only possible missing values emerging from the merging process. At the end of this week I should have a single clean dataset from which I can begin further digging into the relationships and high level significance of the features.

Week 3 goals

1) Do exploratory analysis/preliminary testing to see what features are significant or could be cut from the model
2) See if the data needs to be standardized or feature scaling needs to occur
3) Split the data into training and test datasets

I will do some high-level regression testing to see if there are some features that are entirely insignificant or appear to be too similar to others. I will split the team and year columns as they are not needed for actual testing/regression. They will be rejoined at the end of the project when I need them to represent

Most of the features are on a similar scale with averages or on a per game level. There may be a few features that if I decide to keep I may need to scale to be more in line with the rest of the data(ex: strength of schedule is a ranking of 1-350 to show relative difficulty of schedules for teams compared with others)

I will split the team and year columns as they are not needed for actual testing/regression. They will be rejoined at the end of the project when I need them to identify the predicted winners of matchups based on the projected score differential. I will make sure to keep the correct order of the teams and years in the tuples in the training and test datasets before I split them off so that I will keep the correct order when I rejoin them.

Week 4 goals

1) Test different models and adjust hyperparameters where needed
2) Possible stacking of methods to get an aggregate/average if needed for further confidence

I will use the sklearn data science and machine learning packages to test various models and fine tune the hyperparameters. I will compare predictability across the models. If there are multiple models that I feel comfortable with or it seems that the models are giving significantly different results, then I will use stacking to come to aggregate predictions.

Week 5 goals

1) Settle on a model(s)/method(s) and use visualizations to explain performance of model(s)

The predictions I find from my selected model will be evaluated using metrics and scores found in the sklearn library. I will compare my selected model to the baseline model of selecting the favorite in each model. I will use visualizations through either Python or Tableau to display the comparisons in predictions and scores.

Week 6 goals

1) Build further visualizations to aid presentation/explanations
2) Create video walkthrough
3) Submit project

The presentation should be straightforward where I will have most of my process described in comments or code in my Jupyter Notebook. I will use video recording software on my computer to show myself walking through my Notebook or whatever other relevant files/visualizations.

Week 7 goals

1) Flex week in case I get behind schedule

Delivery plan

In my presentation I will explain my source data and then walk through my Jupyter Notebook to explain my process. I will use visualizations to aid in explaining any decisions or results. Any work that is ready to be shared or can be used when I ask questions will be shared in the "Ready" folder within the shared Google Drive folder. I will use an "In Progress" folder for my personal use for anything I'm currently working on and then paste ready to be shared copies in the "Ready" folder.

*Revised Below Garrett 3/18

I will first explain the process of the gathering of my source data. The difficulties behind it and why I chose those specific sources. Then I will transition to bringing that data into my Jupyter Notebook.

From there I will walk through the Notebook explaining my process. I will first show the steps I took to prepare the data. How I went about feature selection and scaling. Once I have the data I will show the benchmarks and performance metrics that I will be using to evaluate my model.

For the sake of time I will either skip through the runtime or have the results of running the model run before the walkthrough. I'll explain my justification and reasons for choosing/editing the model in this step.

After the model is run I will bring back those performance metrics and benchmarks to compare to and use visualizations to provide more clarity on these comparisons. If there are any visualizations that are not created directly in the Notebook, then I will paste them into a single Word Document that will have a link or comment to show where they are relevant in the Notebook.

Flask will be used to put the model into practice where the user will be given the option to choose a year(within the range of data used) and as long as the 2 inputted teams were in the tournament in the given year, then a predicted winner will be provided.

Any work that is ready to be shared or can be used when I ask questions will be shared in the "Ready" folder within the shared Google Drive folder. I will use an "In Progress" folder for my personal use for anything I'm currently working on and then paste ready to be shared copies in the "Ready" folder.

DTSC 691
Machine Learning
Project Submission

Garrett Fanning

Goals of the project

The National Collegiate Basketball Association (NCAA) has over 350 colleges all competing to earn a spot in the NCAA tournament, with the ultimate goal of winning the tournament. Only 68 colleges are able to play well enough to earn a spot in the tournament. The lowest 8 teams play against each other to whittle down the field to 64 teams, from which a typical bracket style tournament can be created.

The tournament begins in March with games being played nearly on a daily basis through the entire month. The amount of games, exciting finishes, and surprising outcomes/upsets has led this whole experience to be termed as “March Madness”. Fans join in on this experience by trying to guess/predict how the entire tournament will play out. It’s popular for people to create groups with their family, friends, and coworkers to see who was able to be the most accurate with their picks.

For the most part, picking the winner of each game comes down to guessing or using your general basketball knowledge where it’s impossible to consistently predict games correctly. The NCAA tournament at first glance may appear to have a lot of randomness, which has led many people to leverage the vast amount of college basketball data out there to create algorithms that can have more predictability than the simple guessing or just picking the consensus favorite for each game.

Kaggle has many competitions each year to give those with a data science background the opportunity to test their skills trying to create a model to accurately predict any possible matchup between teams in the tournament. There’s plenty of other sponsored opportunities across the Internet for the general public to participate, such as when Warren Buffett offered $1 billion to anyone able to get a perfect bracket, which is virtually impossible. For more context here is a link explaining that opportunity:
<https://bleacherreport.com/articles/1931210-warren-buffet-will-pay-1-billion-to-fan-with-perfect-march-madness-bracket>

My goal is to build a model with better predictability than benchmarks if I were to simply pick the favorite in each matchup(lower seed, #1 seed is better than #2) or pick the team with the better record. Once I am able to train and test a model, I hope to be able to use that model to help me with the current tournament that is about to begin.

I used regular season data to help predict and see how that translates to success in the NCAA tournament. All of the features I will be using are averages or on a per game basis because all teams don't play an equal amount of games in the regular season. Many of these features will not be used if they do not appear significant or are too similar to other features. I will predict winners of matchups in the NCAA tournament through predicting a final score differential, which can be interpreted as a positive differential means one team wins or negative would mean the other team.

I will use the Flask application to show my model in action and the predictions it provides based on the test data.

Data description

1) pip install pandas sklearn sportsreference → in terminal
   from sportsreference.ncaab.teams import Teams

The above code is showing that within sklearn there is an api that I can use to pull college basketball data from a vast sports data repository.

This dataset has 10 years of data with give or take 350 teams of season data per year.Some teams were added in the time period of the data resulting in an uneven amount of teams with data per year. For each team looking at a single year, I collected 40 variables of data that were either statistical game averages or rankings/ratings they were given or earned in that same year. The csv file that I downloaded from the API that directly pulls data from the sportsreference website was 101KB.

There is a glossary explaining each column of data here:
https://www.sports-reference.com/cbb/about/glossary.html

Features pulled from data source(All features or used for merging):
'year','name','abbreviation', 'assist_percentage', 'block_percentage',
'effective_field_goal_percentage', 'field_goal_percentage', 'free_throw_attempt_rate',
'free_throw_percentage', 'free_throws_per_field_goal_attempt', 'offensive_rating',
'offensive_rebound_percentage', 'opp_assist_percentage', 'opp_block_percentage',
'opp_effective_field_goal_percentage','opp_field_goal_percentage',
'opp_free_throw_attempt_rate', 'opp_free_throw_percentage',
'opp_free_throws_per_field_goal_attempt', 'opp_offensive_rating',
'opp_offensive_rebound_percentage','opp_steal_percentage',
'opp_three_point_attempt_rate','opp_three_point_field_goal_percentage',
'opp_two_point_field_goal_percentage','opp_total_rebound_percentage',
'opp_true_shooting_percentage', 'opp_turnover_percentage',

'pace', 'simple_rating_system', 'steal_percentage',
'strength_of_schedule', 'three_point_attempt_rate',
'three_point_field_goal_percentage', 'two_point_field_goal_percentage', 'two_point_field_goals',
'total_rebound_percentage', 'true_shooting_percentage', 'turnover_percentage','win_percentage'

2) The second source is a dataset that has tournament games dating back to 1985. For the range of data used in the models, only games from 2010-2019 are relevant. The only feature from this file are the seeds and the response variable of final score difference is from this dataset. The full dataset downloaded from the website has 2205 rows and 10 columns of data. The csv file size is 101KB.
   The csv was found on:
   <https://data.world/michaelaroy/ncaa-tournament-results/workspace/file?filename=Big_Dance_CSV.csv>
   You can either make an account on the site to download the csv or access the copy I placed in the shared Google Drive folder.

   Data pulled from Source:
   'FinalScore_Difference' (Response), 'Seed' (Feature)
   'Year' and 'Team' (Merging)

Software

I worked in Jupyter notebooks using the necessary data science or machine learning python packages. I didn't need any external database software because in my notebook I'll be pulling in my data directly from the sources. I mainly used packages from sklearn for training and testing various models/methods. Any visualizations were directly in the notebook. I chose Flask as the application to deploy my machine learning model and show it in use. I will use Google Drive for sharing any relevant files or code.

Analyses & Model Specifications

Week 1 schedule

1) Researched topic ideas and submit project proposal
2) Find data sources that would make interesting project topics feasible
   a) There were many paid sources with extensive usable data for my sports topic, but not too many free sources. Fortunately a free source(sportsreference) I found online had all of the data I needed and even an API available through sklearn. The other source with the tournament data took some searching through the internet before I was able to find someone using that csv file for a similar project.

Week 2 schedule

1) Successfully imported data into Jupyter Notebook
    a) Running the API took some time to make sure I was getting the proper data. Each run takes 5-10 minutes, so after multiple attempts I was able to get the right data to save as a csv. Now the notebook only needs to quickly import that csv instead of running the csv.
2) Merge and clean data
    a) Rename the columns to more identifiable names
    b) There were no missing values other than an entire empty column that I removed
    c) A significant amount of time was spent trying to adjust the team names in both datasets to ensure they would merge appropriately

Week 3 schedule

1) Did exploratory analysis/preliminary testing to see what features are significant or could be cut from the model
    a) Did linear regression on all columns/potential features to see significance
    b) Ran correlation matrix to see potential to remove too similar features
    c) Broke down data into subsets of offensive and defensive statistics for a deeper look
2) Split the data into training and test datasets
3) See if the data needs to be standardized or feature scaling needs to occur
    a) Used standard scaler to scale all numerical features

I split the team and year columns as they are not needed for actual testing/regression. They will be rejoined at the end of the project when I need them to identify the predicted winners of matchups based on the projected score differential. I will make sure to keep the correct order of the teams and years in the tuples in the training and test datasets before I split them off so that I will keep the correct order when I rejoin them.

Week 4 goals

1) Use PCA as an alternative method with a reduced feature set
2) Create benchmarks to compare final models to
    a) Created linear regression models where 1) record as only feature 2) seed as only feature
    b) Will compare MSE of those models with final models to see if those simple models were more accurate than the final models I created
3) Tested different models and adjust hyperparameters where needed
    a) Used GridSearch for models to find optimal hyperparameters. With the multiple methods and variation of methods used(scaled data,PCA data) this process took a significant amount of processing time

Week 5 goals

1) Fit the models
2) Find MSE for all models
3) Compare all models and make a decision on chosen/best model
    a) The `Gradient Boosting Model` with no scaling was the lowest MSE and is the model I have chosen. The PCA for that non-scaled model didn't seem to make a signifcant change in the feature set.
    b) The other individual models, according to the MSE, performed very similarly using their respective optimal parameters when fitting. The Voting Regression Model didn't perform as well as I initially thought it would. Upon further reflection that makes sense to me because the individual model optimal parameters and predictions may not actually combine well with the other models to produce a more optimal prediction, with many times the differing models potentially pulling and skewing the results in multiple directions for each prediction.
    c) I was able to show that the individual models I ran, based only on MSE, would be beneficial to use over the benchmark strategies when filling out a tournament bracket to only select the team with the better seed or record.
    d) The features and distribution of those features didn't need to be scaled or narrowed using PCA, and so there wasn't any clear benefit reflected in the MSE using those processes. As a result, the later models I created and fitted I decided that those processes didn't need to be included as they didn't add any clear benefit.

Week 6 goals

1) Create Flask Application
2) Set up test dataset to be easily inputted into application
3) Confirm Flask results are same as results ran directly in Notebook

Week 7 goals

1) Prepare Notebook/Final Materials to be submitted (Project Submission,PDF)
2) Create video walkthrough
    a) Straightforward presentation where I walk through my Jupyter Notebook and show any external sources of data
    b) Show the use of the Flask Application
3) Submit project

Deliverables

1) Video Presentation/Walkthrough

I first explained the process of the gathering of my source data. The difficulties behind it and why I chose those specific sources. Then I transitioned to bringing that data into my Jupyter Notebook.

From there I walk through the Notebook explaining my process. I first showed the steps I took to prepare the data. How I went about feature selection and scaling. Once I have the data I will show the benchmarks and performance metrics that I will be using to evaluate my models.

For the sake of time I either skipped through the runtime or had the results of running the models before the walkthrough.

After the model is run I will bring back those performance metrics and benchmarks to compare to. I then explained my chosen final model using the performance metric.

2) Flask Files

Flask was used to put the model into practice where a user can input the feature data for a specific matchup then a predicted winner and how much that team will win by will be provided. Any necessary external files to run the application will be shared in the Google Drive.

3) Project Submission/PDF

These files will help to summarize and provide all of the relevant files/processes used. They will also be provided in the shared Google Drive.