The topic in this project is about anime acclaim. We want to know whether the sales drop since COVID-19 have been affecting the anime ratings and the number of audience.

# Explore and Clean Data

In [66]:
```
# Update version for Time Series analysis
!pip install statsmodels --upgrade
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (0.13.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from statsmodels) (1.21.5)
Requirement already satisfied: scipy>=1.3 in /usr/local/lib/python3.7/dist-packages (from statsmodels) (1.4.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packages (from statsmodels) (21.3)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages (from statsmodels) (0.5.2)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.7/dist-packages (from statsmodels) (1.3.5)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=21.3->statsmode
ls) (3.0.8)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25->statsmodels) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25->statsmodels)
(2.8.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy>=0.5.2->statsmodels) (1.15.0)
```

In [67]:
```
# Do not show warnings
import warnings
warnings.filterwarnings("ignore")
```

In [68]:
```
# Import scraped data. The scraping is done with Web Scraper as a Google Chrome extension.
# The process took 9 hours.
import pandas as pd
df = pd.read_csv("project_mal.csv")
df_clean = df.copy()
```

## General Info

In [69]:
```
# Data sample
df.head()
```

Out[69]:

| | web-scraper-order | web-scraper-start-url | season | season-href | anime | |
|---|---|---|---|---|---|---|
| 0 | 1648530274-1492 | https://myanimelist.net/anime/season/archive | Spring 2004 | https://myanimelist.net/anime/season/2004/spring | Morizo to Kikkoro | https://myanimelist.net/anime |
| 1 | 1648528165-1160 | https://myanimelist.net/anime/season/archive | Fall 2001 | https://myanimelist.net/anime/season/2001/fall | Hikaru no Go | https://myanimelist.net/anim |
| 2 | 1648547113-4194 | https://myanimelist.net/anime/season/archive | Spring 2018 | https://myanimelist.net/anime/season/2018/spring | Caligula (TV) | https://myanimelist.net/anim |
| 3 | 1648535227-2280 | https://myanimelist.net/anime/season/archive | Spring 2009 | https://myanimelist.net/anime/season/2009/spring | Shin Mazinger Shougeki! Z-hen | https://myanimelist.net/anime |
| 4 | 1648535899-2387 | https://myanimelist.net/anime/season/archive | Summer 2010 | https://myanimelist.net/anime/season/2010/summer | Digimon Xros Wars | https://myanimelist.net/anime |

In [70]:
```
# Data information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5026 entries, 0 to 5025
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   web-scraper-order     5026 non-null   object 
 1   web-scraper-start-url  5026 non-null   object 
 2   season                5026 non-null   object 
 3   season-href           5026 non-null   object 
 4   anime                 5026 non-null   object 
 5   anime-href            5026 non-null   object 
 6   name                  5026 non-null   object 
 7   score                 4080 non-null   float64
 8   rank                  4939 non-null   object 
 9   popularity            5026 non-null   object 
 10  members               5026 non-null   object 
 11  season_repeat         5026 non-null   object 
 12  type                  5026 non-null   object 
 13  studio                4340 non-null   object 
dtypes: float64(1), object(13)
memory usage: 549.8+ KB
```

## Score

In [71]:
```python
# Show anime with highest rating on MyAnimeList
score_rank = df[["anime", "score"]].sort_values(["score"], ascending=False)
score_rank.head(20)
```

Out[71]:

| | anime | score |
|---|---|---|
| 2874 | Fullmetal Alchemist: Brotherhood | 9.15 |
| 3121 | Steins;Gate | 9.09 |
| 4255 | Gintama? | 9.09 |
| 3955 | Shingeki no Kyojin Season 3 Part 2 | 9.08 |
| 1627 | Shingeki no Kyojin: The Final Season Part 2 | 9.06 |
| 3710 | Gintama' | 9.06 |
| 2967 | Hunter x Hunter (2011) | 9.05 |
| 4581 | Fruits Basket: The Final | 9.04 |
| 1391 | Gintama': Enchousen | 9.04 |
| 4659 | Gintama. | 8.99 |
| 2936 | 3-gatsu no Lion 2nd Season | 8.96 |
| 261 | Gintama | 8.95 |
| 2266 | Clannad: After Story | 8.94 |
| 4426 | Code Geass: Hangyaku no Lelouch R2 | 8.91 |
| 3013 | Owarimonogatari 2nd Season | 8.90 |
| 1588 | Kimetsu no Yaiba: Yuukaku-hen | 8.90 |
| 602 | Gintama.: Shirogane no Tamashii-hen - Kouhan-sen | 8.89 |
| 1520 | Shingeki no Kyojin: The Final Season | 8.87 |
| 4813 | Monster | 8.82 |
| 4665 | Gintama.: Shirogane no Tamashii-hen | 8.82 |

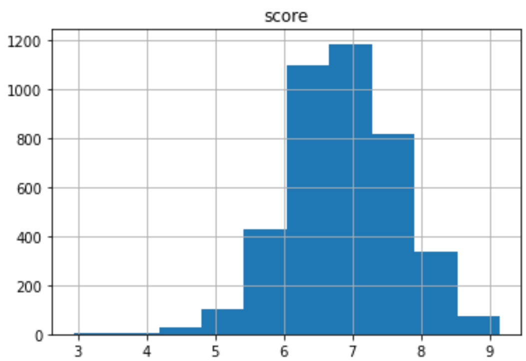
In [72]:
```python
# Rating distribution information
df.describe()
```

Out[72]:

| | score |
|---|---|
| **count** | 4080.000000 |
| **mean** | 6.867375 |
| **std** | 0.797936 |
| **min** | 2.940000 |
| **25%** | 6.340000 |
| **50%** | 6.870000 |
| **75%** | 7.390000 |
| **max** | 9.150000 |

In [73]:
```python
# Distribution of rating
score_rank.hist()
```

Out[73]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fd5ca94b150>]],
      dtype=object)
```



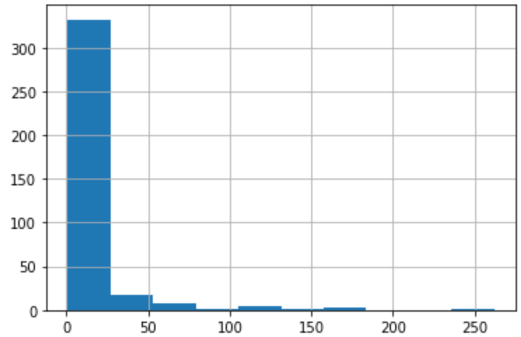

## Studio

In [74]:
```python
# Count numbers of anime made by studios
studio_count = df.groupby(["studio"])["anime"].count().sort_values(ascending=False)
studio_count.head(20)
```

Out[74]:
```
studio
Toei Animation          262
Sunrise                 178
Studio Deen             161
J.C.Staff               160
Madhouse                154
Nippon Animation        122
TMS Entertainment       114
Studio Pierrot          108
Tatsunoko Production    106
OLM                     103
A-1 Pictures             98
Production I.G           77
Gonzo                    73
Xebec                    72
DLE                      66
Bones                    61
Shin-Ei Animation        56
SILVER LINK.             56
Doga Kobo                55
Satelight                52
Name: anime, dtype: int64
```

In [75]:
```python
# Distribution of animes produced by studio
studio_count.hist()
```

Out[75]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd5ca688850>
```




# Ranking

In [76]:
```
# Some features has a "#" prefix; delete them
def delete_sharp(x):
  x = str(x)
  if x == "nan":
    return -1
  elif x[0] == "#":
    return int(x[1:])
  else:
    return x
```

In [77]:
```
# Clean ranking features and show highest rated animes. Null values not shown.
# Some rank are missing because they are not TV shows.
df_clean["rank"] = df_clean["rank"].apply(delete_sharp).astype(int)
ranking = df_clean[["anime", "rank"]][df_clean["rank"] > 0].sort_values("rank")
ranking.head(20)
```

Out[77]:

| | anime | rank |
|---|---|---|
| **2874** | Fullmetal Alchemist: Brotherhood | 1 |
| **4255** | Gintama? | 2 |
| **3121** | Steins;Gate | 3 |
| **3955** | Shingeki no Kyojin Season 3 Part 2 | 4 |
| **1627** | Shingeki no Kyojin: The Final Season Part 2 | 5 |
| **3710** | Gintama' | 6 |
| **2967** | Hunter x Hunter (2011) | 8 |
| **4581** | Fruits Basket: The Final | 9 |
| **1391** | Gintama': Enchousen | 10 |
| **4659** | Gintama. | 12 |
| **2936** | 3-gatsu no Lion 2nd Season | 13 |
| **261** | Gintama | 16 |
| **2266** | Clannad: After Story | 17 |
| **4426** | Code Geass: Hangyaku no Lelouch R2 | 19 |
| **1588** | Kimetsu no Yaiba: Yuukaku-hen | 20 |
| **3013** | Owarimonogatari 2nd Season | 21 |
| **602** | Gintama.: Shirogane no Tamashii-hen - Kouhan-sen | 22 |
| **1520** | Shingeki no Kyojin: The Final Season | 24 |
| **4813** | Monster | 25 |
| **4665** | Gintama.: Shirogane no Tamashii-hen | 26 |

# Popularity

In [78]:
```
# Clean popularity features and show most popular animes. Null values not shown.
# Some rank are missing because they are not TV shows.
df_clean["popularity"] = df_clean["popularity"].apply(delete_sharp).astype(int)
popularity = df_clean[["anime", "popularity"]][df_clean["popularity"] > 0].sort_values("popularity")
popularity.head(20)
```

Out[78]:

| | anime | popularity |
|---|---|---|
| 1139 | Shingeki no Kyojin | 1 |
| 2210 | Death Note | 2 |
| 2874 | Fullmetal Alchemist: Brotherhood | 3 |
| 2405 | One Punch Man | 4 |
| 2973 | Sword Art Online | 5 |
| 1545 | Boku no Hero Academia | 6 |
| 3390 | Tokyo Ghoul | 7 |
| 220 | Naruto | 8 |
| 1823 | Kimetsu no Yaiba | 9 |
| 2967 | Hunter x Hunter (2011) | 10 |
| 2744 | Shingeki no Kyojin Season 2 | 12 |
| 3121 | Steins;Gate | 13 |
| 2305 | Boku no Hero Academia 2nd Season | 14 |
| 4471 | No Game No Life | 15 |
| 4457 | Naruto: Shippuuden | 16 |
| 3572 | Code Geass: Hangyaku no Lelouch | 17 |
| 4130 | Toradora! | 18 |
| 502 | Noragami | 20 |
| 4692 | Shingeki no Kyojin Season 3 | 21 |
| 2723 | Shigatsu wa Kimi no Uso | 22 |

## Membership

In [79]:
```
# Some features have numbers seperated by comma (e.g., 1,234,500); delete them
def delete_comma(x):
  x = str(x)
  if x == "nan":
    return -1
  return int(x.replace(",", ""))
```

In [80]:
```
# Clean membership features and show animes with the most audience. Null values not shown.
df_clean["members"] = df_clean["members"].apply(delete_comma).astype(int)
membership = df_clean[["anime", "members"]][df_clean["members"] > 0].sort_values("members", ascending=False)
membership.head(20)
```

Out[80]:

| | anime | members |
|---|---|---|
| **1139** | Shingeki no Kyojin | 3340909 |
| **2210** | Death Note | 3331576 |
| **2874** | Fullmetal Alchemist: Brotherhood | 2822351 |
| **2405** | One Punch Man | 2731807 |
| **2973** | Sword Art Online | 2701850 |
| **1545** | Boku no Hero Academia | 2570234 |
| **3390** | Tokyo Ghoul | 2436823 |
| **220** | Naruto | 2426258 |
| **1823** | Kimetsu no Yaiba | 2339213 |
| **2967** | Hunter x Hunter (2011) | 2301610 |
| **2744** | Shingeki no Kyojin Season 2 | 2249125 |
| **3121** | Steins;Gate | 2192048 |
| **2305** | Boku no Hero Academia 2nd Season | 2156929 |
| **4471** | No Game No Life | 2116096 |
| **4457** | Naruto: Shippuuden | 2060041 |
| **3572** | Code Geass: Hangyaku no Lelouch | 1948413 |
| **4130** | Toradora! | 1936811 |
| **502** | Noragami | 1895744 |
| **4692** | Shingeki no Kyojin Season 3 | 1892855 |
| **2723** | Shigatsu wa Kimi no Uso | 1887627 |

## Seasons

In [81]:

```
# Count number of animes for each season and order them from most to least
season_count = df.groupby(["season"])["anime"].count().sort_values(ascending=False)
season_count.head(20)
```

Out[81]:

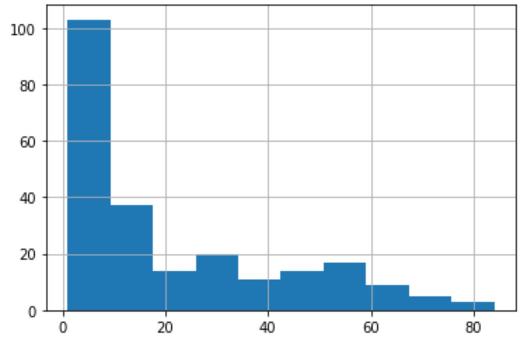
```
season
Spring 2017    84
Fall 2016      80
Spring 2018    76
Spring 2016    75
Spring 2006    72
Spring 2014    72
Fall 2017      70
Spring 2011    70
Fall 2018      66
Fall 2015      65
Spring 2015    64
Summer 2016    63
Spring 2020    63
Spring 2021    62
Summer 2015    61
Winter 2021    61
Spring 2007    60
Spring 2013    59
Summer 2018    59
Summer 2017    59
Name: anime, dtype: int64
```

In [82]:

```
# Distribution of animes in a season
season_count.hist()
```

Out[82]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd63788be50>
```




# Preprocess Data

## Into Time Series

In [83]:
```
# Only select relevant features
df_time = df_clean[["season", "anime", "score", "rank", "popularity", "members"]]
df_time.head()
```

Out[83]:

| | season | anime | score | rank | popularity | members |
|---|---|---|---|---|---|---|
| **0** | Spring 2004 | Morizo to Kikkoro | NaN | 12665 | 16458 | 205 |
| **1** | Fall 2001 | Hikaru no Go | 8.08 | 461 | 1442 | 123926 |
| **2** | Spring 2018 | Caligula (TV) | 6.01 | 8722 | 2105 | 71817 |
| **3** | Spring 2009 | Shin Mazinger Shougeki! Z-hen | 7.67 | 1165 | 4785 | 14062 |
| **4** | Summer 2010 | Digimon Xros Wars | 6.68 | 5296 | 2765 | 44255 |

In [84]:
```
# Drop null scores and null membership. (Will be used to calculate acclaim feature)
df_time = df_time.dropna(subset = ["score", "members"])
df_time.head()
```

Out[84]:

| | season | anime | score | rank | popularity | members |
|---|---|---|---|---|---|---|
| **1** | Fall 2001 | Hikaru no Go | 8.08 | 461 | 1442 | 123926 |
| **2** | Spring 2018 | Caligula (TV) | 6.01 | 8722 | 2105 | 71817 |
| **3** | Spring 2009 | Shin Mazinger Shougeki! Z-hen | 7.67 | 1165 | 4785 | 14062 |
| **4** | Summer 2010 | Digimon Xros Wars | 6.68 | 5296 | 2765 | 44255 |
| **5** | Spring 1967 | Ribbon no Kishi | 6.71 | 5156 | 7710 | 4262 |

In [85]:
```
# Order anime seasons from oldest (low) to newest (high). (1917 has the oldest anime listed in MyAnimeList)
def get_time(x):
  season, year = x.split(" ")[0], int(x.split(" ")[1])
  season_order = {"Winter": 0, "Spring": 1, "Summer": 2, "Fall": 3}
  return (year - 1917) * 4 + season_order[season]
```

In [86]:
```
# Check order of anime seasons
if ("time" not in df_time.columns):
  df_time.insert(0, "time", df["season"].apply(get_time))
df_time[["time", "season"]].head(10)
```

Out[86]:

| | time | season |
|---|---|---|
| **1** | 339 | Fall 2001 |
| **2** | 405 | Spring 2018 |
| **3** | 369 | Spring 2009 |
| **4** | 374 | Summer 2010 |
| **5** | 201 | Spring 1967 |
| **7** | 378 | Summer 2011 |
| **8** | 373 | Spring 2010 |
| **9** | 351 | Fall 2004 |
| **10** | 349 | Spring 2004 |
| **11** | 327 | Fall 1998 |

In [87]:

```
# Shows number of anime in each season.
# For cross-checking with MyAnimeList to see if the scraped data is complete.
pd.set_option("display.max_rows", 1000)
check_group = df_time.groupby("time")
df_check = pd.DataFrame(data={"season": check_group["season"].first(), "count": check_group["anime"].count()})
print(df_check)
pd.reset_option("display.max_rows")
```

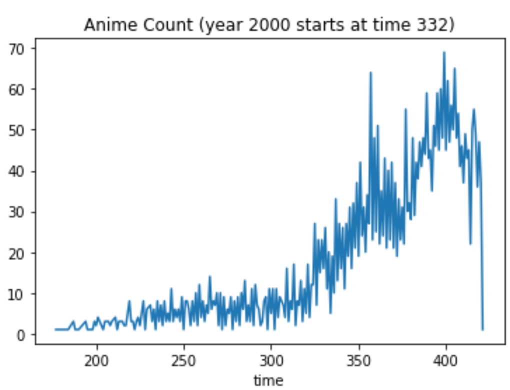
| time | season | count |
|---|---|---|
| 177 | Spring 1961 | 1 |
| 182 | Summer 1962 | 1 |
| 184 | Winter 1963 | 1 |
| 187 | Fall 1963 | 3 |
| 188 | Winter 1964 | 1 |
| 189 | Spring 1964 | 1 |
| 190 | Summer 1964 | 1 |
| 194 | Summer 1965 | 3 |
| 195 | Fall 1965 | 1 |
| 196 | Winter 1966 | 1 |
| 197 | Spring 1966 | 1 |
| 198 | Summer 1966 | 1 |
| 199 | Fall 1966 | 3 |
| 200 | Winter 1967 | 2 |
| 201 | Spring 1967 | 4 |
| 204 | Winter 1968 | 1 |
| 205 | Spring 1968 | 3 |
| 207 | Fall 1968 | 3 |
| 208 | Winter 1969 | 2 |
| 209 | Spring 1969 | 3 |
| 211 | Fall 1969 | 4 |
| 212 | Winter 1970 | 1 |
| 213 | Spring 1970 | 3 |
| 215 | Fall 1970 | 3 |
| 216 | Winter 1971 | 2 |
| 217 | Spring 1971 | 2 |
| 219 | Fall 1971 | 8 |
| 220 | Winter 1972 | 3 |
| 221 | Spring 1972 | 3 |
| 222 | Summer 1972 | 1 |
| 223 | Fall 1972 | 3 |
| 224 | Winter 1973 | 4 |
| 225 | Spring 1973 | 2 |
| 227 | Fall 1973 | 8 |
| 228 | Winter 1974 | 1 |
| 229 | Spring 1974 | 6 |
| 231 | Fall 1974 | 7 |
| 232 | Winter 1975 | 3 |
| 233 | Spring 1975 | 6 |
| 234 | Summer 1975 | 1 |
| 235 | Fall 1975 | 8 |
| 236 | Winter 1976 | 3 |
| 237 | Spring 1976 | 7 |
| 238 | Summer 1976 | 2 |
| 239 | Fall 1976 | 8 |
| 240 | Winter 1977 | 3 |
| 241 | Spring 1977 | 5 |
| 242 | Summer 1977 | 3 |
| 243 | Fall 1977 | 11 |
| 244 | Winter 1978 | 3 |
| 245 | Spring 1978 | 6 |
| 246 | Summer 1978 | 4 |
| 247 | Fall 1978 | 6 |
| 248 | Winter 1979 | 3 |
| 249 | Spring 1979 | 9 |
| 250 | Summer 1979 | 1 |
| 251 | Fall 1979 | 8 |
| 252 | Winter 1980 | 8 |
| 253 | Spring 1980 | 6 |
| 254 | Summer 1980 | 2 |
| 255 | Fall 1980 | 8 |
| 256 | Winter 1981 | 3 |
| 257 | Spring 1981 | 10 |
| 258 | Summer 1981 | 2 |
| 259 | Fall 1981 | 12 |
| 260 | Winter 1982 | 4 |
| 261 | Spring 1982 | 8 |
| 262 | Summer 1982 | 3 |
| 263 | Fall 1982 | 7 |
| 264 | Winter 1983 | 5 |
| 265 | Spring 1983 | 14 |
| 266 | Summer 1983 | 6 |
| 267 | Fall 1983 | 8 |
| 268 | Winter 1984 | 7 |
| 269 | Spring 1984 | 10 |
| 270 | Summer 1984 | 2 |
| 271 | Fall 1984 | 10 |
| 272 | Winter 1985 | 1 |
| 273 | Spring 1985 | 9 |
| 274 | Summer 1985 | 2 |
| 275 | Fall 1985 | 6 |
| 276 | Winter 1986 | 5 |
| 277 | Spring 1986 | 9 |
| 278 | Summer 1986 | 1 |

| | | | |
|---|---|---|---|
| 279 | Fall | 1986 | 8 |
| 280 | Winter | 1987 | 3 |
| 281 | Spring | 1987 | 9 |
| 282 | Summer | 1987 | 2 |
| 283 | Fall | 1987 | 10 |
| 284 | Winter | 1988 | 6 |
| 285 | Spring | 1988 | 13 |
| 286 | Summer | 1988 | 3 |
| 287 | Fall | 1988 | 7 |
| 288 | Winter | 1989 | 3 |
| 289 | Spring | 1989 | 11 |
| 290 | Summer | 1989 | 2 |
| 291 | Fall | 1989 | 12 |
| 292 | Winter | 1990 | 7 |
| 293 | Spring | 1990 | 6 |
| 294 | Summer | 1990 | 2 |
| 295 | Fall | 1990 | 3 |
| 296 | Winter | 1991 | 8 |
| 297 | Spring | 1991 | 9 |
| 298 | Summer | 1991 | 1 |
| 299 | Fall | 1991 | 11 |
| 300 | Winter | 1992 | 5 |
| 301 | Spring | 1992 | 11 |
| 302 | Summer | 1992 | 1 |
| 303 | Fall | 1992 | 11 |
| 304 | Winter | 1993 | 4 |
| 305 | Spring | 1993 | 9 |
| 307 | Fall | 1993 | 7 |
| 308 | Winter | 1994 | 4 |
| 309 | Spring | 1994 | 16 |
| 310 | Summer | 1994 | 3 |
| 311 | Fall | 1994 | 8 |
| 312 | Winter | 1995 | 6 |
| 313 | Spring | 1995 | 17 |
| 314 | Summer | 1995 | 2 |
| 315 | Fall | 1995 | 8 |
| 316 | Winter | 1996 | 7 |
| 317 | Spring | 1996 | 13 |
| 318 | Summer | 1996 | 3 |
| 319 | Fall | 1996 | 11 |
| 320 | Winter | 1997 | 5 |
| 321 | Spring | 1997 | 17 |
| 322 | Summer | 1997 | 4 |
| 323 | Fall | 1997 | 12 |
| 324 | Winter | 1998 | 12 |
| 325 | Spring | 1998 | 27 |
| 326 | Summer | 1998 | 7 |
| 327 | Fall | 1998 | 23 |
| 328 | Winter | 1999 | 15 |
| 329 | Spring | 1999 | 23 |
| 330 | Summer | 1999 | 16 |
| 331 | Fall | 1999 | 26 |
| 332 | Winter | 2000 | 11 |
| 333 | Spring | 2000 | 20 |
| 334 | Summer | 2000 | 5 |
| 335 | Fall | 2000 | 19 |
| 336 | Winter | 2001 | 10 |
| 337 | Spring | 2001 | 33 |
| 338 | Summer | 2001 | 13 |
| 339 | Fall | 2001 | 27 |
| 340 | Winter | 2002 | 16 |
| 341 | Spring | 2002 | 26 |
| 342 | Summer | 2002 | 11 |
| 343 | Fall | 2002 | 27 |
| 344 | Winter | 2003 | 19 |
| 345 | Spring | 2003 | 31 |
| 346 | Summer | 2003 | 16 |
| 347 | Fall | 2003 | 32 |
| 348 | Winter | 2004 | 21 |
| 349 | Spring | 2004 | 37 |
| 350 | Summer | 2004 | 19 |
| 351 | Fall | 2004 | 42 |
| 352 | Winter | 2005 | 24 |
| 353 | Spring | 2005 | 31 |
| 354 | Summer | 2005 | 20 |
| 355 | Fall | 2005 | 34 |
| 356 | Winter | 2006 | 27 |
| 357 | Spring | 2006 | 64 |
| 358 | Summer | 2006 | 23 |
| 359 | Fall | 2006 | 48 |
| 360 | Winter | 2007 | 25 |
| 361 | Spring | 2007 | 51 |
| 362 | Summer | 2007 | 22 |
| 363 | Fall | 2007 | 35 |
| 364 | Winter | 2008 | 24 |
| 365 | Spring | 2008 | 43 |

```
366   Summer 2008     21
367     Fall 2008     40
368   Winter 2009     23
369   Spring 2009     42
370   Summer 2009     21
371     Fall 2009     37
372   Winter 2010     19
373   Spring 2010     33
374   Summer 2010     23
375     Fall 2010     31
376   Winter 2011     22
377   Spring 2011     55
378   Summer 2011     30
379     Fall 2011     32
380   Winter 2012     28
381   Spring 2012     48
382   Summer 2012     29
383     Fall 2012     42
384   Winter 2013     38
385   Spring 2013     47
386   Summer 2013     41
387     Fall 2013     48
388   Winter 2014     44
389   Spring 2014     59
390   Summer 2014     43
391     Fall 2014     45
392   Winter 2015     35
393   Spring 2015     51
394   Summer 2015     46
395     Fall 2015     59
396   Winter 2016     45
397   Spring 2016     60
398   Summer 2016     48
399     Fall 2016     69
400   Winter 2017     45
401   Spring 2017     62
402   Summer 2017     47
403     Fall 2017     56
404   Winter 2018     50
405   Spring 2018     65
406   Summer 2018     48
407     Fall 2018     54
408   Winter 2019     41
409   Spring 2019     46
410   Summer 2019     37
411     Fall 2019     49
412   Winter 2020     43
413   Spring 2020     45
414   Summer 2020     22
415     Fall 2020     50
416   Winter 2021     55
417   Spring 2021     49
418   Summer 2021     36
419     Fall 2021     47
420   Winter 2022     37
421   Spring 2022      1
```

In [88]:
```
# Plot numbers of anime per season
df_check["count"].plot(title="Anime Count (year 2000 starts at time 332)")
```

Out[88]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd5ca50ac50>
```




## Calculate Acclaim

In [89]:
```
# Transform anime membership feature for analysis.
# The scale of anime membership should be logarithmic.
```

```
# This is because it will only take a few quality improvement for membership to become tenfold.
from math import log10
if ("members_log" not in df_time.columns):
  df_time.insert(7, "members_log", df_time["members"].apply(lambda x: log10(x)))
df_time.head()
```

Out[89]:

| | time | season | anime | score | rank | popularity | members | members_log |
|---|---|---|---|---|---|---|---|---|
| **1** | 339 | Fall 2001 | Hikaru no Go | 8.08 | 461 | 1442 | 123926 | 5.093162 |
| **2** | 405 | Spring 2018 | Caligula (TV) | 6.01 | 8722 | 2105 | 71817 | 4.856227 |
| **3** | 369 | Spring 2009 | Shin Mazinger Shougeki! Z-hen | 7.67 | 1165 | 4785 | 14062 | 4.148047 |
| **4** | 374 | Summer 2010 | Digimon Xros Wars | 6.68 | 5296 | 2765 | 44255 | 4.645962 |
| **5** | 201 | Spring 1967 | Ribbon no Kishi | 6.71 | 5156 | 7710 | 4262 | 3.629613 |

In [90]:

```
# Prepare to scale members feature.
# The chosen training data for the scale to fit is in between 2000 and 2017, inclusive.
df_member_train = pd.DataFrame(data={"members": df_time[(df_time["time"] >= 332) & (df_time["time"] < 404)]["members_log"]})
print(df_member_train.head())
df_member_all = pd.DataFrame(data={"members": df_time["members_log"]})
print(df_member_all.head())
```

```
    members
1  5.093162
3  4.148047
4  4.645962
7  4.932174
8  4.766086
    members
1  5.093162
2  4.856227
3  4.148047
4  4.645962
5  3.629613
```

In [91]:

```
# Scale members so that it has the same scale as rating.
# The acclaim feature to be calculated has members and ratings factor.
# Scaling members is important so that the number of members do not overwhelm the rating.
# The training period is 2000-2017, the testing period is 2018-2019, the prediction period is 2020-2021.
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(1, 10))
scaler.fit(df_member_train)
df_time["members_scaled"] = scaler.transform(df_member_all)
df_time
```

Out[91]:

| | time | season | anime | score | rank | popularity | members | members_log | members_scaled |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 339 | Fall 2001 | Hikaru no Go | 8.08 | 461 | 1442 | 123926 | 5.093162 | 6.843900 |
| **2** | 405 | Spring 2018 | Caligula (TV) | 6.01 | 8722 | 2105 | 71817 | 4.856227 | 6.321226 |
| **3** | 369 | Spring 2009 | Shin Mazinger Shougeki! Z-hen | 7.67 | 1165 | 4785 | 14062 | 4.148047 | 4.758995 |
| **4** | 374 | Summer 2010 | Digimon Xros Wars | 6.68 | 5296 | 2765 | 44255 | 4.645962 | 5.857386 |
| **5** | 201 | Spring 1967 | Ribbon no Kishi | 6.71 | 5156 | 7710 | 4262 | 3.629613 | 3.615340 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **5021** | 402 | Summer 2017 | Made in Abyss | 8.68 | 52 | 95 | 1104027 | 6.042980 | 8.939178 |
| **5022** | 248 | Winter 1979 | Hana no Ko Lunlun | 6.87 | 4392 | 7464 | 4663 | 3.668665 | 3.701488 |
| **5023** | 402 | Summer 2017 | Gamers! | 6.81 | 4647 | 295 | 571452 | 5.756980 | 8.308267 |
| **5024** | 352 | Winter 2005 | Lime-iro Ryuukitan X | 5.47 | 10737 | 7794 | 4147 | 3.617734 | 3.589135 |
| **5025** | 387 | Fall 2013 | Chuldong! Super Wings | 5.93 | 9098 | 15340 | 280 | 2.447158 | 1.006868 |

4080 rows × 9 columns

In [92]:

```
# Calculate acclaim feature.
# The acclaim of an anime is defined to be the average rating of the anime multipled by the number of viewers.
# The acclaim will be high if many viewers view it and give it a high rating.
# A perfect rating with minimal viewers will not score as high.
# On the other hand, a bad rating with lots of viewers will also not score as high.
# The members and ratings are both in the scale of 1-10, so they will contribute equally.
if ("acclaim" not in df_time.columns):
  df_time.insert(9, "acclaim", df_time["score"] * df_time["members_scaled"])
df_time
```

Out[92]:

| | time | season | anime | score | rank | popularity | members | members_log | members_scaled | acclaim |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 339 | Fall 2001 | Hikaru no Go | 8.08 | 461 | 1442 | 123926 | 5.093162 | 6.843900 | 55.298711 |
| **2** | 405 | Spring 2018 | Caligula (TV) | 6.01 | 8722 | 2105 | 71817 | 4.856227 | 6.321226 | 37.990566 |
| **3** | 369 | Spring 2009 | Shin Mazinger Shougeki! Z-hen | 7.67 | 1165 | 4785 | 14062 | 4.148047 | 4.758995 | 36.501488 |
| **4** | 374 | Summer 2010 | Digimon Xros Wars | 6.68 | 5296 | 2765 | 44255 | 4.645962 | 5.857386 | 39.127335 |
| **5** | 201 | Spring 1967 | Ribbon no Kishi | 6.71 | 5156 | 7710 | 4262 | 3.629613 | 3.615340 | 24.258934 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **5021** | 402 | Summer 2017 | Made in Abyss | 8.68 | 52 | 95 | 1104027 | 6.042980 | 8.939178 | 77.592061 |
| **5022** | 248 | Winter 1979 | Hana no Ko Lunlun | 6.87 | 4392 | 7464 | 4663 | 3.668665 | 3.701488 | 25.429225 |
| **5023** | 402 | Summer 2017 | Gamers! | 6.81 | 4647 | 295 | 571452 | 5.756980 | 8.308267 | 56.579301 |
| **5024** | 352 | Winter 2005 | Lime-iro Ryuukitan X | 5.47 | 10737 | 7794 | 4147 | 3.617734 | 3.589135 | 19.632567 |
| **5025** | 387 | Fall 2013 | Chuldong! Super Wings | 5.93 | 9098 | 15340 | 280 | 2.447158 | 1.006868 | 5.970726 |

4080 rows × 10 columns

In [93]:

```python
# Check for seasons which have less than 5 animes.
# This check is for averaging the 5 highest acclaim level into a seasonal aggregate.
# Usually, anime seasons are defined by their top performing animes.
# This is because there are many animes with low production values that is not hyped.
# The acclaim level should not depend on how many low ranking animes there are in a season.
df_check[df_check["count"] < 5]
```

Out[93]:

| time | season | count |
|---|---|---|
| **177** | Spring 1961 | 1 |
| **182** | Summer 1962 | 1 |
| **184** | Winter 1963 | 1 |
| **187** | Fall 1963 | 3 |
| **188** | Winter 1964 | 1 |
| **...** | ... | ... |
| **310** | Summer 1994 | 3 |
| **314** | Summer 1995 | 2 |
| **318** | Summer 1996 | 3 |
| **322** | Summer 1997 | 4 |
| **421** | Spring 2022 | 1 |

68 rows × 2 columns

In [94]:

```python
# Create a single-variable time series data based on acclaim.
# The acclaim of a season is the average top 5 acclaims in the season.
groupby = df_time.sort_values("acclaim", ascending=False).groupby("time")
df_group = pd.DataFrame(data={"acclaim": groupby["acclaim"].agg(lambda x: x.head(5).mean())})
df_group.head(10)
```

Out[94]:

| time | acclaim |
|---|---|
| 177 | 11.656072 |
| 182 | 10.353630 |
| 184 | 31.968613 |
| 187 | 16.077200 |
| 188 | 11.801606 |
| 189 | 13.202998 |
| 190 | 11.843116 |
| 194 | 11.648020 |
| 195 | 26.838891 |
| 196 | 21.477920 |

In [95]:
```
# Smooth the time series with moving average of 4 periods.
# The members and ratings of an anime in MyAnimeList has some high peaks.
# This is due to the existence of top rated one-hit wonder animes that overall boost the season's acclaim.
# To smoothen this peak, moving average is used, and the value now represents an average acclaim for the past year.
series_smooth = df_group.loc[329:]["acclaim"]
df_smooth = pd.DataFrame(data={"acclaim": series_smooth.rolling(4).mean()})
df_smooth.head(10)
```

Out[95]:

| time | acclaim |
|---|---|
| 329 | NaN |
| 330 | NaN |
| 331 | NaN |
| 332 | 47.205416 |
| 333 | 49.153404 |
| 334 | 45.934765 |
| 335 | 44.680575 |
| 336 | 45.791237 |
| 337 | 44.808614 |
| 338 | 49.243936 |

## Exploring Time Series

In [96]:
```
# Seperate training data from 2000 to 2017, inclusive.
# The data will be divided into training, testing, and prediction data.
# The training data is for modelling.
# The testing data is for validating the model.
# Finally, the prediction data is used to compare with real data to see any deviations from the validated model.
df_train = df_smooth.loc[332:403]
df_train
```

Out[96]:

| | acclaim |
|---|---|
| **time** | |
| **332** | 47.205416 |
| **333** | 49.153404 |
| **334** | 45.934765 |
| **335** | 44.680575 |
| **336** | 45.791237 |
| **...** | ... |
| **399** | 71.686477 |
| **400** | 71.065487 |
| **401** | 70.257343 |
| **402** | 69.510881 |
| **403** | 69.486766 |

72 rows × 1 columns

In [97]:

```
# Seperate testing data from 2018 to 2019, inclusive.
df_test = df_smooth.loc[404:411]
df_test
```

Out[97]:

| | acclaim |
|---|---|
| **time** | |
| **404** | 69.722358 |
| **405** | 69.865659 |
| **406** | 70.468917 |
| **407** | 70.689232 |
| **408** | 72.135727 |
| **409** | 73.217926 |
| **410** | 72.911791 |
| **411** | 71.643476 |

In [98]:

```
# Seperate prediction data from 2020 to 2021, inclusive. (COVID-19 period)
df_predict = df_smooth.loc[412:419]
df_predict
```

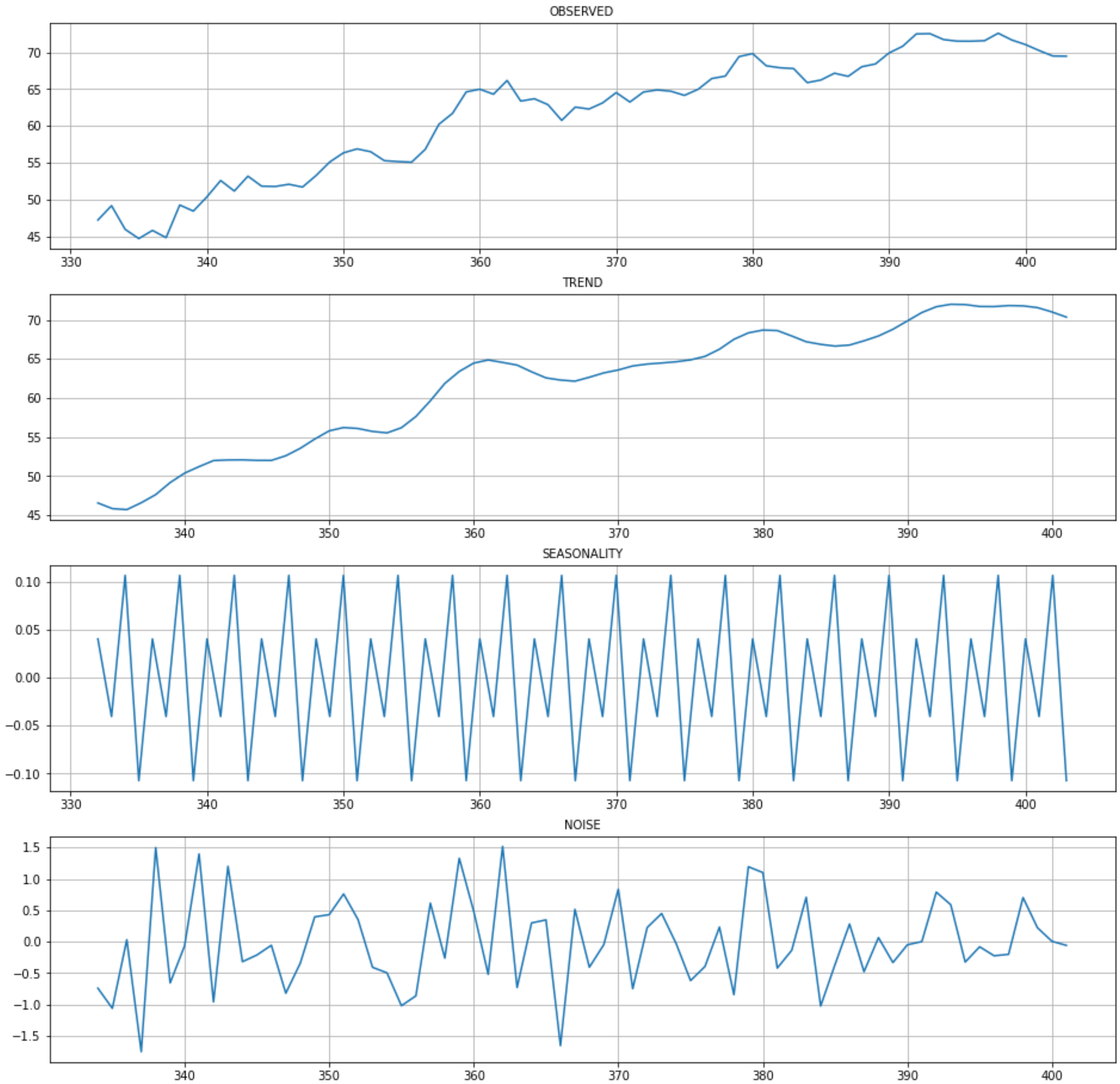
Out[98]:

| | acclaim |
|---|---|
| **time** | |
| **412** | 68.695018 |
| **413** | 66.549721 |
| **414** | 65.906475 |
| **415** | 66.581891 |
| **416** | 68.998947 |
| **417** | 69.878765 |
| **418** | 68.608553 |
| **419** | 68.510468 |

In [99]:

```
# Decompose time series for analysis
import statsmodels.api as sm
import matplotlib.pyplot as plt

res = sm.tsa.seasonal_decompose(df_train["acclaim"], period=4)
fig, axs = plt.subplots(4, figsize=(16,16))
axs[0].set_title("OBSERVED", fontsize=10)
axs[0].plot(res.observed)
axs[0].grid()

axs[1].set_title("TREND", fontsize=10)
axs[1].plot(res.trend)
```

```
axs[1].grid()

axs[2].set_title("SEASONALITY", fontsize=10)
axs[2].plot(res.seasonal)
axs[2].grid()

axs[3].set_title("NOISE", fontsize=10)
axs[3].plot(res.resid)
axs[3].grid()

plt.show()
```



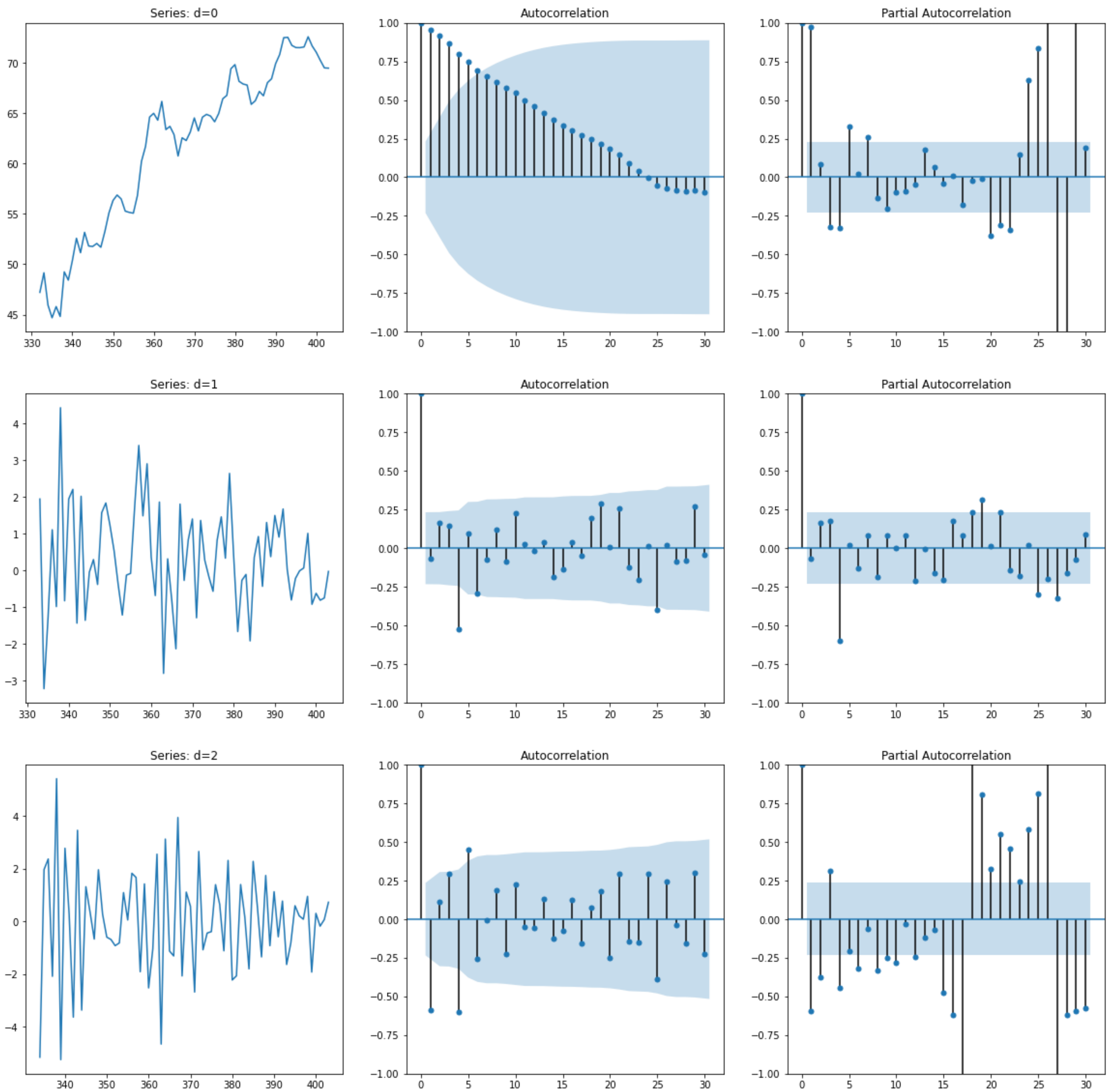

# Model Data (ARIMA)

## Model Creation

In [100…

```
# Plot time series and their correlation functions
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Original Series
fig, axes = plt.subplots(3, 3, figsize=(20, 20))
axes[0, 0].plot(df_train)
axes[0, 0].set_title("Series: d=0")
plot_acf(df_train["acclaim"], ax=axes[0, 1], lags=30)
plot_pacf(df_train["acclaim"], ax=axes[0, 2], lags=30)
```

```python
# 1st Differencing
axes[1, 0].plot(df_train["acclaim"].diff())
axes[1, 0].set_title("Series: d=1")
plot_acf(df_train["acclaim"].diff().dropna(), ax=axes[1, 1], lags=30)
plot_pacf(df_train["acclaim"].diff().dropna(), ax=axes[1, 2], lags=30)

# 2nd Differencing
axes[2, 0].plot(df_train["acclaim"].diff().diff())
axes[2, 0].set_title("Series: d=2")
plot_acf(df_train["acclaim"].diff().diff().dropna(), ax=axes[2, 1], lags=30)
plot_pacf(df_train["acclaim"].diff().diff().dropna(), ax=axes[2, 2], lags=30)

plt.show()
```




In [101…

```python
# Fit ARIMA(4,1,4).
# From the plotted data, d=0 is not stationary and has an increasing trend.
# d=2's ACF plot is too negative for q=1 due to over-differencing.
# d=1 is chosen, and their corresponding p and q is 4 and 4, from PACF and ACF, respectively.
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(df_train["acclaim"],order=(4,1,4))
result = model.fit()
result.summary()
```

Out[101]:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | acclaim | **No. Observations:** | 72 |
| **Model:** | ARIMA(4, 1, 4) | **Log Likelihood** | -106.690 |
| **Date:** | Thu, 14 Apr 2022 | **AIC** | 231.380 |
| **Time:** | 09:47:36 | **BIC** | 251.744 |
| **Sample:** | 0 | **HQIC** | 239.478 |
| | - 72 | | |
| **Covariance Type:** | opg | | |

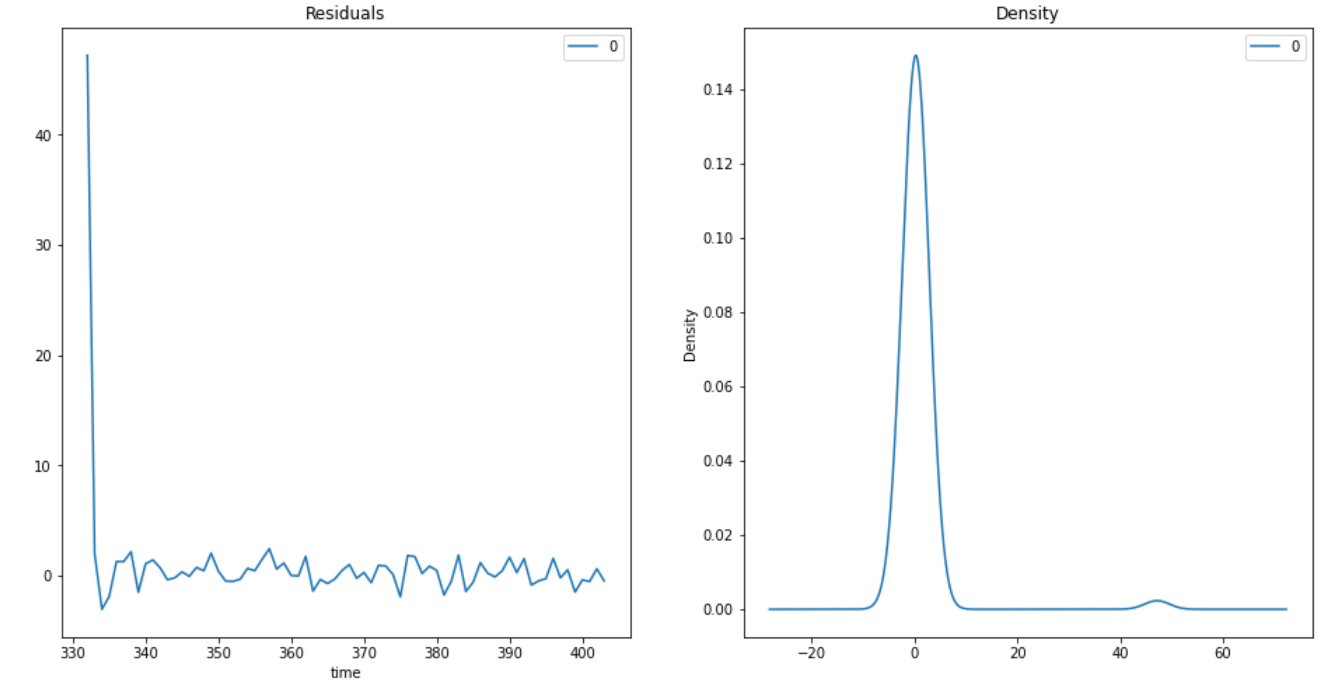
| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | -0.2389 | 0.359 | -0.666 | 0.505 | -0.942 | 0.464 |
| **ar.L2** | 0.2339 | 0.327 | 0.715 | 0.475 | -0.407 | 0.875 |
| **ar.L3** | 0.0795 | 0.195 | 0.408 | 0.683 | -0.302 | 0.461 |
| **ar.L4** | -0.4520 | 0.153 | -2.948 | 0.003 | -0.753 | -0.151 |
| **ma.L1** | 0.3656 | 0.371 | 0.986 | 0.324 | -0.361 | 1.092 |
| **ma.L2** | 0.1978 | 0.402 | 0.492 | 0.623 | -0.591 | 0.986 |
| **ma.L3** | 0.3525 | 0.356 | 0.990 | 0.322 | -0.345 | 1.050 |
| **ma.L4** | -0.0964 | 0.332 | -0.291 | 0.771 | -0.746 | 0.554 |
| **sigma2** | 1.1383 | 0.236 | 4.832 | 0.000 | 0.677 | 1.600 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.01 | **Jarque-Bera (JB):** | 0.90 |
| **Prob(Q):** | 0.93 | **Prob(JB):** | 0.64 |
| **Heteroskedasticity (H):** | 0.78 | **Skew:** | -0.15 |
| **Prob(H) (two-sided):** | 0.55 | **Kurtosis:** | 2.54 |

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [102…

```
# Plot residual errors.
# To ensure that all trends and seasonality are captured in the model, residuals are inspected.
# The first residual is due to first-order differencing.
# There seems to be no pattern within the residual.
residuals = pd.DataFrame(result.resid)
fig, ax = plt.subplots(1, 2, figsize=(16, 8))
residuals.plot(title="Residuals", ax=ax[0])
residuals.plot(kind='kde', title='Density', ax=ax[1])
plt.show()
```




## Training Data

In [103…

```
# Predict acclaim. The fist value is zero due to differencing.
df_train["acclaim_pred"] = result.predict()
df_train.head(10)
```

Out[103]:

| time | acclaim | acclaim_pred |
|---|---|---|
| 332 | 47.205416 | 0.000000 |
| 333 | 49.153404 | 47.205408 |
| 334 | 45.934765 | 49.005994 |
| 335 | 44.680575 | 46.584295 |
| 336 | 45.791237 | 44.530184 |
| 337 | 44.808614 | 43.548526 |
| 338 | 49.243936 | 47.093241 |
| 339 | 48.419327 | 49.951515 |
| 340 | 50.368392 | 49.310926 |
| 341 | 52.579151 | 51.172408 |

In [104…

```
# Measure model training performance. MAPE is less than 5%, but testing performance must also be checked.
from sklearn import metrics

acclaim_actual = df_train["acclaim"].loc[333:]
acclaim_pred = df_train["acclaim_pred"].loc[333:]

print('(MAE) Mean Absolute Error is:\n', metrics.mean_absolute_error(acclaim_actual, acclaim_pred), '\n')
print('(MSE) Mean Squared Error is:\n', metrics.mean_squared_error(acclaim_actual, acclaim_pred), '\n')
print('(MAPE) Mean Absolute Percentage Error is:\n',
      metrics.mean_absolute_percentage_error(acclaim_actual, acclaim_pred), '\n')
print('(R^2) Coefficient of Determination is:\n', metrics.r2_score(acclaim_actual, acclaim_pred), '\n')
```

```
(MAE) Mean Absolute Error is:
 0.9019361693723408

(MSE) Mean Squared Error is:
 1.2678215342044463

(MAPE) Mean Absolute Percentage Error is:
 0.015285779188376996

(R^2) Coefficient of Determination is:
 0.9802651667406256
```

## Testing Data

In [105…

```
# Predict acclaim for years 2018 and 2019
df_test_result = pd.DataFrame(data={
    "acclaim": df_test["acclaim"].values.tolist(),
    "acclaim_pred": result.predict(start=72, end=79).tolist()
})
df_test_result
```

Out[105]:

| | acclaim | acclaim_pred |
|---|---|---|
| 0 | 69.722358 | 69.320048 |
| 1 | 69.865659 | 69.826807 |
| 2 | 70.468917 | 69.771160 |
| 3 | 70.689232 | 69.948018 |
| 4 | 72.135727 | 70.008409 |
| 5 | 73.217926 | 69.801864 |
| 6 | 72.911791 | 69.904539 |
| 7 | 71.643476 | 69.756570 |

In [106…

```
# Measure model training performance. MAPE is less than 5%, so the model is totally acceptable.
from sklearn import metrics

acclaim_test_actual = df_test_result["acclaim"]
acclaim_test_pred = df_test_result["acclaim_pred"]

print('(MAE) Mean Absolute Error is:\n', metrics.mean_absolute_error(acclaim_test_actual, acclaim_test_pred), '\n')
print('(MSE) Mean Squared Error is:\n', metrics.mean_squared_error(acclaim_test_actual, acclaim_test_pred), '\n')
print('(MAPE) Mean Absolute Percentage Error is:\n',
      metrics.mean_absolute_percentage_error(acclaim_test_actual, acclaim_test_pred), '\n')
print('(R^2) Coefficient of Determination is:\n', metrics.r2_score(acclaim_test_actual, acclaim_test_pred), '\n')
```

```
(MAE) Mean Absolute Error is:
 1.5397088577939808

(MSE) Mean Squared Error is:
 3.7498210764823647

(MAPE) Mean Absolute Percentage Error is:
 0.021305314501417497

(R^2) Coefficient of Determination is:
 -1.3630930744177152
```

## Prediction Data

In [107…

```
# Predict acclaim for years 2020 and 2021
df_predict_result = pd.DataFrame(data={
    "acclaim": df_predict["acclaim"].values.tolist(),
    "acclaim_pred": result.predict(start=80, end=87).tolist()
})
df_predict_result
```

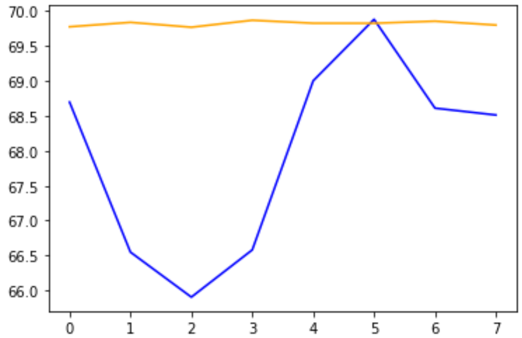
Out[107]:

| | acclaim | acclaim_pred |
|---|---|---|
| 0 | 68.695018 | 69.772208 |
| 1 | 66.549721 | 69.835389 |
| 2 | 65.906475 | 69.765780 |
| 3 | 66.581891 | 69.865310 |
| 4 | 68.998947 | 69.823211 |
| 5 | 69.878765 | 69.822452 |
| 6 | 68.608553 | 69.852164 |
| 7 | 68.510468 | 69.796554 |

In [108…

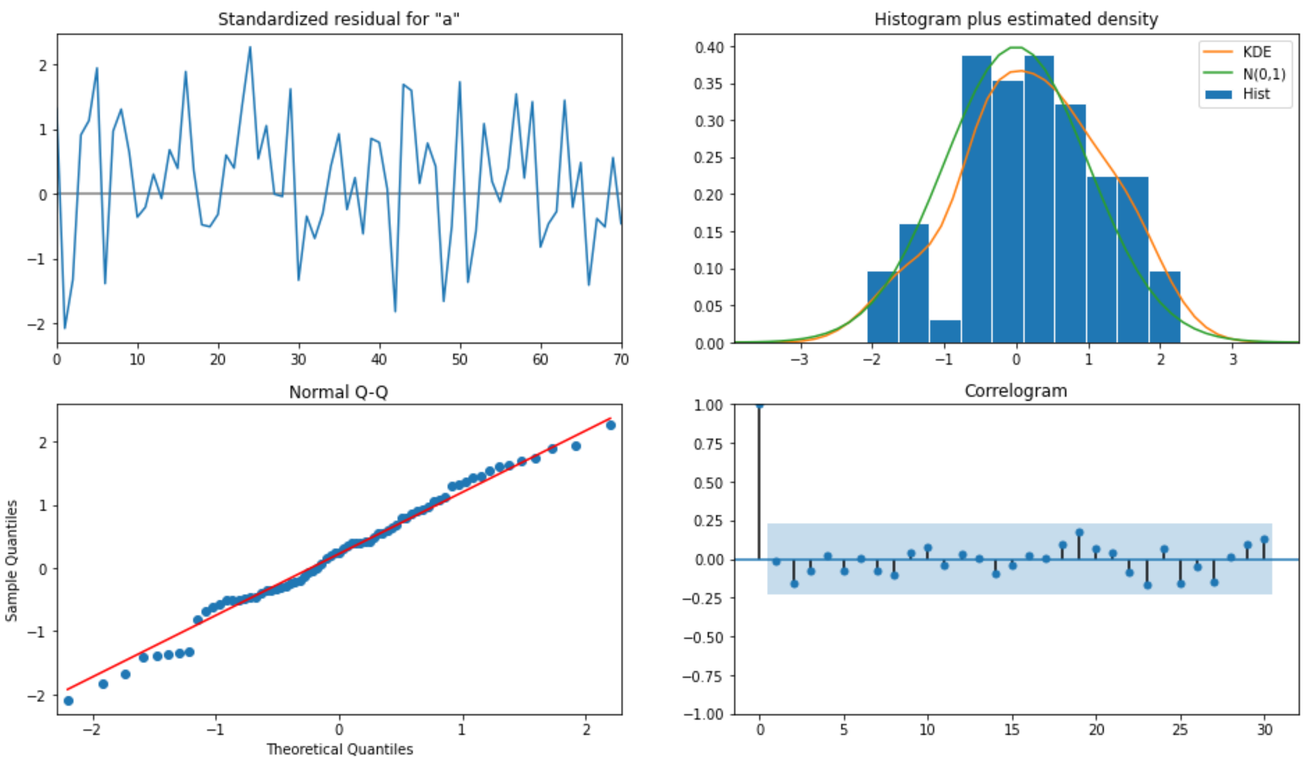
```
# Plot actual acclaim against the predicted values.
# Performance is not measured here since the model is already validated during testing prediction.
acclaim_predict_actual = df_predict_result["acclaim"]
acclaim_predict_pred = df_predict_result["acclaim_pred"]
```

```
acclaim_predict_actual.plot(color="blue")
acclaim_predict_pred.plot(color="orange")
plt.show()
```



In [109…

```
# Model diagnostics
result.plot_diagnostics(figsize=(16,9), lags=30)
plt.show()
```




In [110…

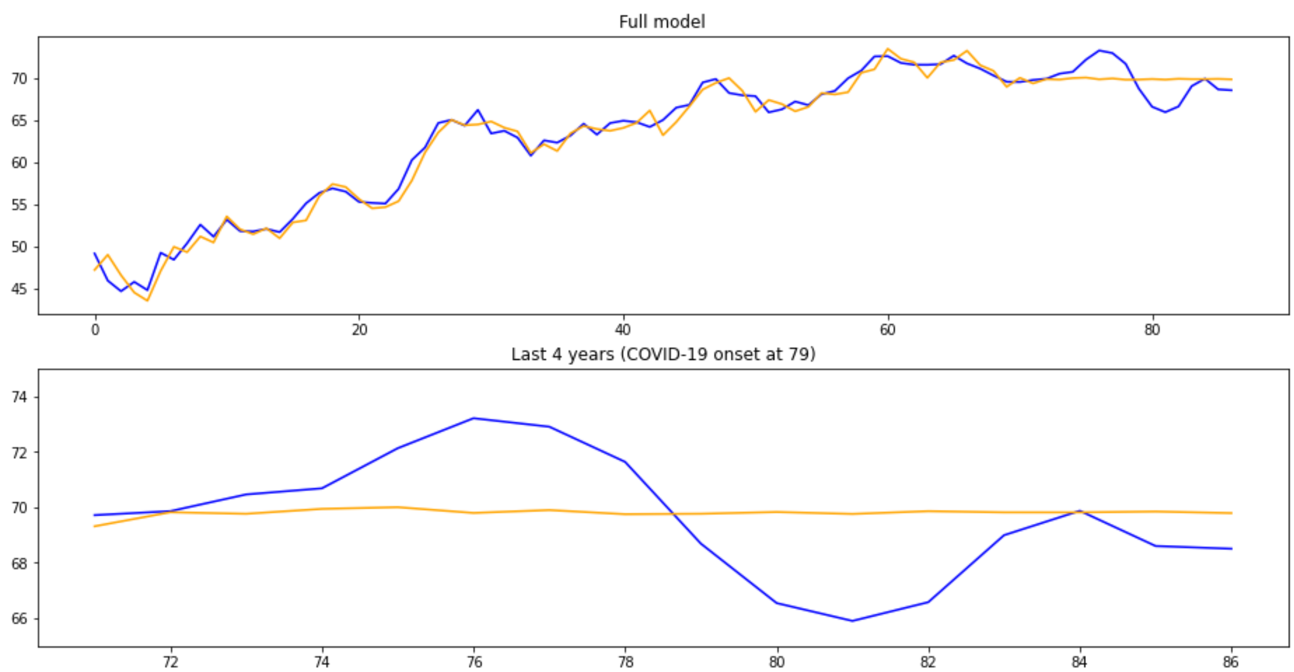
```
# Full prediction result (2000-2021)
df_full = df_smooth.loc[333:419]
df_full_result = pd.DataFrame(data={
    "acclaim": df_full["acclaim"].values.tolist(),
    "acclaim_pred": result.predict(start=1, end=87).tolist()
})
df_full_result.tail(16)
```

Out[110]:

| | acclaim | acclaim_pred |
|---|---|---|
| **71** | 69.722358 | 69.320048 |
| **72** | 69.865659 | 69.826807 |
| **73** | 70.468917 | 69.771160 |
| **74** | 70.689232 | 69.948018 |
| **75** | 72.135727 | 70.008409 |
| **76** | 73.217926 | 69.801864 |
| **77** | 72.911791 | 69.904539 |
| **78** | 71.643476 | 69.756570 |
| **79** | 68.695018 | 69.772208 |
| **80** | 66.549721 | 69.835389 |
| **81** | 65.906475 | 69.765780 |
| **82** | 66.581891 | 69.865310 |
| **83** | 68.998947 | 69.823211 |
| **84** | 69.878765 | 69.822452 |
| **85** | 68.608553 | 69.852164 |
| **86** | 68.510468 | 69.796554 |

In [111…

```
# Plot actual and predicted acclaim values
fig, axes = plt.subplots(2, figsize=(16, 8))

df_full_result["acclaim"].plot(color="blue", ax=axes[0], title="Full model")
df_full_result["acclaim_pred"].plot(color="orange", ax=axes[0])

df_full_result["acclaim"].loc[71:86].plot(color="blue", ax=axes[1], title="Last 4 years (COVID-19 onset at 79)")
df_full_result["acclaim_pred"].loc[71:86].plot(color="orange", ax=axes[1])
axes[1].set_ylim([65, 75])

plt.show()
```




From the model, we see that ARIMA always give the same prediction after the training period.

This is because the ARIMA model depends on the existence of the previous timesteps in order to calculate moving average.

Since the train data available to the model is at most 2017, predicting anime acclaim for 2021 is unreasonable.

In other words, ARIMA is a single step model, it can only predict one timestep in the future.

Therefore, we will disregard this model and use a multi step model, which is LSTM.

# Model Data (LSTM)

## Further Preprocessing

In [112…

```
# Eliminate the randomness for reproducibility
import os
os.environ["PYTHONHASHSEED"] = "0"
os.environ["TF_DETERMINISTIC_OPS"] = "0"
os.environ["CUDA_VISBLE_DEVICE"] = ""

import random
random.seed(0)

import numpy as np
np.random.seed(0)

import tensorflow as tf
tf.random.set_seed(0)
```

In [113…

```
# Capture all acclaim data from 2000 to 2021
df_model = df_smooth.loc[332:419].reset_index()
df_model = df_model.drop("time", axis=1)
df_model
```

Out[113]:

| | acclaim |
|---|---|
| **0** | 47.205416 |
| **1** | 49.153404 |
| **2** | 45.934765 |
| **3** | 44.680575 |
| **4** | 45.791237 |
| **...** | ... |
| **83** | 66.581891 |
| **84** | 68.998947 |
| **85** | 69.878765 |
| **86** | 68.608553 |
| **87** | 68.510468 |

88 rows × 1 columns

In [114…

```
# Scale acclaim feature.
# The LSTM model will initialize the kernel as a zero value.
# If the acclaim feature is scaled to be from 0 to 1, the LSTM will fit the data faster.
# The training data from 2000-2017 is fit to the scaler, then all data is transformed.
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(df_model[0:71])

df_scaled = df_model.copy()
df_scaled["acclaim"] = scaler.transform(df_model)
df_scaled
```

Out[114]:

| | acclaim |
|---|---|
| 0 | 0.090414 |
| 1 | 0.160171 |
| 2 | 0.044912 |
| 3 | 0.000000 |
| 4 | 0.039772 |
| ... | ... |
| 83 | 0.784279 |
| 84 | 0.870833 |
| 85 | 0.902339 |
| 86 | 0.856853 |
| 87 | 0.853341 |

88 rows × 1 columns

In [115…

```
# Seperate training, testing, and prediction data
df_train = df_scaled[0:72]
df_test = df_scaled[64:80]
df_predict = df_scaled[72:88]
df_predict
```

Out[115]:

| | acclaim |
|---|---|
| 72 | 0.896738 |
| 73 | 0.901870 |
| 74 | 0.923472 |
| 75 | 0.931362 |
| 76 | 0.983160 |
| 77 | 1.021913 |
| 78 | 1.010951 |
| 79 | 0.965533 |
| 80 | 0.859949 |
| 81 | 0.783127 |
| 82 | 0.760093 |
| 83 | 0.784279 |
| 84 | 0.870833 |
| 85 | 0.902339 |
| 86 | 0.856853 |
| 87 | 0.853341 |

## LSTM Input Processing

In [116…

```
# Create a window class.
# This class specifies the input and output values of LSTM.
# Input width is the number of input timesteps.
# Label width is the number of output timesteps.
# Shift is the number of timesteps between the last input and the last output.
class WindowGenerator():
  def __init__(self, input_width, label_width, shift):
    self.input_width = input_width
    self.label_width = label_width
    self.shift = shift

    self.total_window_size = input_width + shift

    self.input_slice = slice(0, input_width)
    self.labels_slice = slice(self.total_window_size - self.label_width, None)
```

In [117…

```
# Create a split function.
# This function splits the dataset into input and labels for output.
def split_window(self, features):
```

```
  inputs = features[:, self.input_slice, :]
  labels = features[:, self.labels_slice, :]

  inputs.set_shape([None, self.input_width, None])
  labels.set_shape([None, self.label_width, None])

  return inputs, labels

WindowGenerator.split_window = split_window
```

In [118…

```
# Create dataset generation function.
# LSTM models require input to be 3-dimensional.
# The dimensions are batch size, timesteps, and features.
# The generation is based upon a Keras utility function.
import numpy as np
import tensorflow as tf

BATCH_SIZE = 32

def make_dataset(self, data):
  data = np.array(data, dtype=np.float32)
  ds = tf.keras.utils.timeseries_dataset_from_array(
      data=data,
      targets=None,
      sequence_length=self.total_window_size,
      sequence_stride=1,
      batch_size=BATCH_SIZE,
      seed=0)

  ds = ds.map(self.split_window)

  return ds

WindowGenerator.make_dataset = make_dataset
```

## Model Creation

In [119…

```
# Create LSTM model.
# There will be 32 node in this LSTM.
# The model predicts the next 8 timesteps given the previous 8 timesteps and its short-term memory.
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import LSTM, Dense, Reshape
from tensorflow.python.keras.initializers import zeros

STEPS = 8
UNITS = 32

model = Sequential()
model.add(LSTM(UNITS, return_sequences=False))
model.add(Dense(STEPS, kernel_initializer=zeros()))
model.add(Reshape([STEPS, 1]))
```

In [120…

```
# Create an object that can create LSTM dataset with specified configuration.
window = WindowGenerator(input_width=STEPS, label_width=STEPS, shift=STEPS)
```

In [121…

```
# Fit the LSTM model to training data, running for 200 epochs.
MAX_EPOCHS = 200

model.compile(loss='mse', optimizer='adam', metrics=['mape'])
history = model.fit(window.make_dataset(df_train), epochs=MAX_EPOCHS)
```

```
Epoch 1/200
2/2 [==============================] - 3s 16ms/step - loss: 0.5129 - mape: 99.8244
Epoch 2/200
2/2 [==============================] - 0s 16ms/step - loss: 0.5046 - mape: 98.9564
Epoch 3/200
2/2 [==============================] - 0s 17ms/step - loss: 0.4956 - mape: 98.0169
Epoch 4/200
2/2 [==============================] - 0s 16ms/step - loss: 0.4856 - mape: 96.9688
Epoch 5/200
2/2 [==============================] - 0s 16ms/step - loss: 0.4744 - mape: 95.7909
Epoch 6/200
2/2 [==============================] - 0s 16ms/step - loss: 0.4619 - mape: 94.4577
Epoch 7/200
2/2 [==============================] - 0s 14ms/step - loss: 0.4478 - mape: 92.9377
Epoch 8/200
2/2 [==============================] - 0s 15ms/step - loss: 0.4319 - mape: 91.1928
Epoch 9/200
2/2 [==============================] - 0s 15ms/step - loss: 0.4137 - mape: 89.1759
Epoch 10/200
2/2 [==============================] - 0s 16ms/step - loss: 0.3931 - mape: 86.8294
Epoch 11/200
2/2 [==============================] - 0s 19ms/step - loss: 0.3696 - mape: 84.0831
Epoch 12/200
2/2 [==============================] - 0s 20ms/step - loss: 0.3428 - mape: 80.8519
Epoch 13/200
2/2 [==============================] - 0s 18ms/step - loss: 0.3125 - mape: 77.0337
Epoch 14/200
2/2 [==============================] - 0s 19ms/step - loss: 0.2786 - mape: 72.5093
Epoch 15/200
2/2 [==============================] - 0s 18ms/step - loss: 0.2410 - mape: 67.1439
Epoch 16/200
2/2 [==============================] - 0s 21ms/step - loss: 0.2004 - mape: 60.7933
Epoch 17/200
2/2 [==============================] - 0s 22ms/step - loss: 0.1579 - mape: 53.3181
Epoch 18/200
2/2 [==============================] - 0s 19ms/step - loss: 0.1156 - mape: 44.6099
Epoch 19/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0763 - mape: 34.6967
Epoch 20/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0434 - mape: 24.3551
Epoch 21/200
2/2 [==============================] - 0s 14ms/step - loss: 0.0205 - mape: 16.4163
Epoch 22/200
2/2 [==============================] - 0s 15ms/step - loss: 0.0097 - mape: 13.6000
Epoch 23/200
2/2 [==============================] - 0s 15ms/step - loss: 0.0104 - mape: 17.3274
Epoch 24/200
2/2 [==============================] - 0s 17ms/step - loss: 0.0187 - mape: 23.4413
Epoch 25/200
2/2 [==============================] - 0s 15ms/step - loss: 0.0282 - mape: 28.7895
Epoch 26/200
2/2 [==============================] - 0s 15ms/step - loss: 0.0335 - mape: 31.5157
Epoch 27/200
2/2 [==============================] - 0s 17ms/step - loss: 0.0328 - mape: 31.1816
Epoch 28/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0279 - mape: 28.6701
Epoch 29/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0216 - mape: 25.2535
Epoch 30/200
2/2 [==============================] - 0s 18ms/step - loss: 0.0162 - mape: 21.8275
Epoch 31/200
2/2 [==============================] - 0s 15ms/step - loss: 0.0128 - mape: 18.9738
Epoch 32/200
2/2 [==============================] - 0s 14ms/step - loss: 0.0114 - mape: 16.9209
Epoch 33/200
2/2 [==============================] - 0s 17ms/step - loss: 0.0113 - mape: 15.7794
Epoch 34/200
2/2 [==============================] - 0s 17ms/step - loss: 0.0117 - mape: 15.1750
Epoch 35/200
2/2 [==============================] - 0s 18ms/step - loss: 0.0120 - mape: 14.8230
Epoch 36/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0119 - mape: 14.5393
Epoch 37/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0115 - mape: 14.2811
Epoch 38/200
2/2 [==============================] - 0s 18ms/step - loss: 0.0107 - mape: 14.0696
Epoch 39/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0100 - mape: 13.9584
Epoch 40/200
2/2 [==============================] - 0s 17ms/step - loss: 0.0093 - mape: 13.9934
Epoch 41/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0090 - mape: 14.2235
Epoch 42/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0089 - mape: 14.5782
Epoch 43/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0090 - mape: 14.9318
```

```
Epoch 44/200
2/2 [==============================] - 0s 28ms/step - loss: 0.0092 - mape: 15.2029
Epoch 45/200
2/2 [==============================] - 0s 17ms/step - loss: 0.0093 - mape: 15.3166
Epoch 46/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0092 - mape: 15.2388
Epoch 47/200
2/2 [==============================] - 0s 18ms/step - loss: 0.0090 - mape: 14.9920
Epoch 48/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0088 - mape: 14.6306
Epoch 49/200
2/2 [==============================] - 0s 15ms/step - loss: 0.0085 - mape: 14.2275
Epoch 50/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0083 - mape: 13.8280
Epoch 51/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0081 - mape: 13.4559
Epoch 52/200
2/2 [==============================] - 0s 18ms/step - loss: 0.0080 - mape: 13.1461
Epoch 53/200
2/2 [==============================] - 0s 17ms/step - loss: 0.0079 - mape: 12.8992
Epoch 54/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0078 - mape: 12.7201
Epoch 55/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0077 - mape: 12.5968
Epoch 56/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0076 - mape: 12.5182
Epoch 57/200
2/2 [==============================] - 0s 16ms/step - loss: 0.0075 - mape: 12.4733
Epoch 58/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0075 - mape: 12.4515
Epoch 59/200
2/2 [==============================] - 0s 21ms/step - loss: 0.0075 - mape: 12.4464
Epoch 60/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0074 - mape: 12.4398
Epoch 61/200
2/2 [==============================] - 0s 21ms/step - loss: 0.0074 - mape: 12.4206
Epoch 62/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0074 - mape: 12.3851
Epoch 63/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0074 - mape: 12.3281
Epoch 64/200
2/2 [==============================] - 0s 26ms/step - loss: 0.0073 - mape: 12.2525
Epoch 65/200
2/2 [==============================] - 0s 26ms/step - loss: 0.0073 - mape: 12.1656
Epoch 66/200
2/2 [==============================] - 0s 36ms/step - loss: 0.0073 - mape: 12.0736
Epoch 67/200
2/2 [==============================] - 0s 36ms/step - loss: 0.0072 - mape: 11.9819
Epoch 68/200
2/2 [==============================] - 0s 29ms/step - loss: 0.0072 - mape: 11.8965
Epoch 69/200
2/2 [==============================] - 0s 44ms/step - loss: 0.0072 - mape: 11.8239
Epoch 70/200
2/2 [==============================] - 0s 34ms/step - loss: 0.0072 - mape: 11.7644
Epoch 71/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0071 - mape: 11.7189
Epoch 72/200
2/2 [==============================] - 0s 42ms/step - loss: 0.0071 - mape: 11.6856
Epoch 73/200
2/2 [==============================] - 0s 35ms/step - loss: 0.0071 - mape: 11.6613
Epoch 74/200
2/2 [==============================] - 0s 18ms/step - loss: 0.0071 - mape: 11.6425
Epoch 75/200
2/2 [==============================] - 0s 37ms/step - loss: 0.0071 - mape: 11.6260
Epoch 76/200
2/2 [==============================] - 0s 30ms/step - loss: 0.0071 - mape: 11.6093
Epoch 77/200
2/2 [==============================] - 0s 24ms/step - loss: 0.0071 - mape: 11.5907
Epoch 78/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0071 - mape: 11.5684
Epoch 79/200
2/2 [==============================] - 0s 17ms/step - loss: 0.0071 - mape: 11.5427
Epoch 80/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0071 - mape: 11.5144
Epoch 81/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0071 - mape: 11.4851
Epoch 82/200
2/2 [==============================] - 0s 24ms/step - loss: 0.0071 - mape: 11.4563
Epoch 83/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0071 - mape: 11.4297
Epoch 84/200
2/2 [==============================] - 0s 39ms/step - loss: 0.0071 - mape: 11.4072
Epoch 85/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0071 - mape: 11.3882
Epoch 86/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0070 - mape: 11.3725
```

```
Epoch 87/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0070 - mape: 11.3597
Epoch 88/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0070 - mape: 11.3491
Epoch 89/200
2/2 [==============================] - 0s 29ms/step - loss: 0.0070 - mape: 11.3397
Epoch 90/200
2/2 [==============================] - 0s 38ms/step - loss: 0.0070 - mape: 11.3308
Epoch 91/200
2/2 [==============================] - 0s 28ms/step - loss: 0.0070 - mape: 11.3218
Epoch 92/200
2/2 [==============================] - 0s 37ms/step - loss: 0.0070 - mape: 11.3126
Epoch 93/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0070 - mape: 11.3029
Epoch 94/200
2/2 [==============================] - 0s 20ms/step - loss: 0.0070 - mape: 11.2927
Epoch 95/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0070 - mape: 11.2825
Epoch 96/200
2/2 [==============================] - 0s 39ms/step - loss: 0.0070 - mape: 11.2727
Epoch 97/200
2/2 [==============================] - 0s 42ms/step - loss: 0.0070 - mape: 11.2635
Epoch 98/200
2/2 [==============================] - 0s 24ms/step - loss: 0.0070 - mape: 11.2553
Epoch 99/200
2/2 [==============================] - 0s 67ms/step - loss: 0.0070 - mape: 11.2482
Epoch 100/200
2/2 [==============================] - 0s 29ms/step - loss: 0.0070 - mape: 11.2423
Epoch 101/200
2/2 [==============================] - 0s 27ms/step - loss: 0.0070 - mape: 11.2372
Epoch 102/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0070 - mape: 11.2328
Epoch 103/200
2/2 [==============================] - 0s 50ms/step - loss: 0.0070 - mape: 11.2289
Epoch 104/200
2/2 [==============================] - 0s 29ms/step - loss: 0.0070 - mape: 11.2252
Epoch 105/200
2/2 [==============================] - 0s 33ms/step - loss: 0.0070 - mape: 11.2216
Epoch 106/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0070 - mape: 11.2179
Epoch 107/200
2/2 [==============================] - 0s 27ms/step - loss: 0.0070 - mape: 11.2141
Epoch 108/200
2/2 [==============================] - 0s 24ms/step - loss: 0.0070 - mape: 11.2104
Epoch 109/200
2/2 [==============================] - 0s 26ms/step - loss: 0.0070 - mape: 11.2068
Epoch 110/200
2/2 [==============================] - 0s 50ms/step - loss: 0.0070 - mape: 11.2034
Epoch 111/200
2/2 [==============================] - 0s 31ms/step - loss: 0.0070 - mape: 11.2003
Epoch 112/200
2/2 [==============================] - 0s 28ms/step - loss: 0.0070 - mape: 11.1975
Epoch 113/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0070 - mape: 11.1950
Epoch 114/200
2/2 [==============================] - 0s 34ms/step - loss: 0.0070 - mape: 11.1929
Epoch 115/200
2/2 [==============================] - 0s 27ms/step - loss: 0.0070 - mape: 11.1909
Epoch 116/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0070 - mape: 11.1891
Epoch 117/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0070 - mape: 11.1873
Epoch 118/200
2/2 [==============================] - 0s 38ms/step - loss: 0.0070 - mape: 11.1856
Epoch 119/200
2/2 [==============================] - 0s 26ms/step - loss: 0.0070 - mape: 11.1840
Epoch 120/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0070 - mape: 11.1823
Epoch 121/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0070 - mape: 11.1807
Epoch 122/200
2/2 [==============================] - 0s 26ms/step - loss: 0.0070 - mape: 11.1791
Epoch 123/200
2/2 [==============================] - 0s 39ms/step - loss: 0.0070 - mape: 11.1776
Epoch 124/200
2/2 [==============================] - 0s 35ms/step - loss: 0.0070 - mape: 11.1762
Epoch 125/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0070 - mape: 11.1749
Epoch 126/200
2/2 [==============================] - 0s 37ms/step - loss: 0.0070 - mape: 11.1737
Epoch 127/200
2/2 [==============================] - 0s 30ms/step - loss: 0.0070 - mape: 11.1725
Epoch 128/200
2/2 [==============================] - 0s 34ms/step - loss: 0.0070 - mape: 11.1715
Epoch 129/200
2/2 [==============================] - 0s 27ms/step - loss: 0.0070 - mape: 11.1705
```

```
Epoch 130/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0070 - mape: 11.1695
Epoch 131/200
2/2 [==============================] - 0s 28ms/step - loss: 0.0070 - mape: 11.1685
Epoch 132/200
2/2 [==============================] - 0s 44ms/step - loss: 0.0070 - mape: 11.1675
Epoch 133/200
2/2 [==============================] - 0s 46ms/step - loss: 0.0070 - mape: 11.1666
Epoch 134/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0070 - mape: 11.1656
Epoch 135/200
2/2 [==============================] - 0s 44ms/step - loss: 0.0070 - mape: 11.1647
Epoch 136/200
2/2 [==============================] - 0s 47ms/step - loss: 0.0070 - mape: 11.1638
Epoch 137/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0070 - mape: 11.1629
Epoch 138/200
2/2 [==============================] - 0s 38ms/step - loss: 0.0070 - mape: 11.1620
Epoch 139/200
2/2 [==============================] - 0s 42ms/step - loss: 0.0070 - mape: 11.1612
Epoch 140/200
2/2 [==============================] - 0s 31ms/step - loss: 0.0070 - mape: 11.1604
Epoch 141/200
2/2 [==============================] - 0s 24ms/step - loss: 0.0070 - mape: 11.1596
Epoch 142/200
2/2 [==============================] - 0s 39ms/step - loss: 0.0070 - mape: 11.1588
Epoch 143/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0070 - mape: 11.1580
Epoch 144/200
2/2 [==============================] - 0s 36ms/step - loss: 0.0070 - mape: 11.1572
Epoch 145/200
2/2 [==============================] - 0s 31ms/step - loss: 0.0070 - mape: 11.1564
Epoch 146/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0070 - mape: 11.1556
Epoch 147/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0070 - mape: 11.1548
Epoch 148/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0070 - mape: 11.1540
Epoch 149/200
2/2 [==============================] - 0s 31ms/step - loss: 0.0070 - mape: 11.1532
Epoch 150/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0070 - mape: 11.1525
Epoch 151/200
2/2 [==============================] - 0s 42ms/step - loss: 0.0070 - mape: 11.1517
Epoch 152/200
2/2 [==============================] - 0s 24ms/step - loss: 0.0070 - mape: 11.1509
Epoch 153/200
2/2 [==============================] - 0s 35ms/step - loss: 0.0070 - mape: 11.1502
Epoch 154/200
2/2 [==============================] - 0s 35ms/step - loss: 0.0070 - mape: 11.1494
Epoch 155/200
2/2 [==============================] - 0s 19ms/step - loss: 0.0070 - mape: 11.1486
Epoch 156/200
2/2 [==============================] - 0s 49ms/step - loss: 0.0070 - mape: 11.1478
Epoch 157/200
2/2 [==============================] - 0s 22ms/step - loss: 0.0070 - mape: 11.1471
Epoch 158/200
2/2 [==============================] - 0s 25ms/step - loss: 0.0070 - mape: 11.1463
Epoch 159/200
2/2 [==============================] - 0s 37ms/step - loss: 0.0070 - mape: 11.1455
Epoch 160/200
2/2 [==============================] - 0s 47ms/step - loss: 0.0070 - mape: 11.1447
Epoch 161/200
2/2 [==============================] - 0s 30ms/step - loss: 0.0070 - mape: 11.1439
Epoch 162/200
2/2 [==============================] - 0s 33ms/step - loss: 0.0070 - mape: 11.1431
Epoch 163/200
2/2 [==============================] - 0s 23ms/step - loss: 0.0070 - mape: 11.1423
Epoch 164/200
2/2 [==============================] - 0s 27ms/step - loss: 0.0070 - mape: 11.1416
Epoch 165/200
2/2 [==============================] - 0s 30ms/step - loss: 0.0070 - mape: 11.1408
Epoch 166/200
2/2 [==============================] - 0s 44ms/step - loss: 0.0070 - mape: 11.1400
Epoch 167/200
2/2 [==============================] - 0s 28ms/step - loss: 0.0070 - mape: 11.1392
Epoch 168/200
2/2 [==============================] - 0s 38ms/step - loss: 0.0070 - mape: 11.1384
Epoch 169/200
2/2 [==============================] - 0s 29ms/step - loss: 0.0070 - mape: 11.1376
Epoch 170/200
2/2 [==============================] - 0s 36ms/step - loss: 0.0070 - mape: 11.1368
Epoch 171/200
2/2 [==============================] - 0s 31ms/step - loss: 0.0070 - mape: 11.1360
Epoch 172/200
2/2 [==============================] - 0s 40ms/step - loss: 0.0069 - mape: 11.1352
```

```
Epoch 173/200
2/2 [==============================] - 0s 30ms/step - loss: 0.0069 - mape: 11.1344
Epoch 174/200
2/2 [==============================] - 0s 38ms/step - loss: 0.0069 - mape: 11.1336
Epoch 175/200
2/2 [==============================] - 0s 30ms/step - loss: 0.0069 - mape: 11.1328
Epoch 176/200
2/2 [==============================] - 0s 60ms/step - loss: 0.0069 - mape: 11.1320
Epoch 177/200
2/2 [==============================] - 0s 63ms/step - loss: 0.0069 - mape: 11.1312
Epoch 178/200
2/2 [==============================] - 0s 44ms/step - loss: 0.0069 - mape: 11.1304
Epoch 179/200
2/2 [==============================] - 0s 42ms/step - loss: 0.0069 - mape: 11.1296
Epoch 180/200
2/2 [==============================] - 0s 40ms/step - loss: 0.0069 - mape: 11.1288
Epoch 181/200
2/2 [==============================] - 0s 31ms/step - loss: 0.0069 - mape: 11.1280
Epoch 182/200
2/2 [==============================] - 0s 42ms/step - loss: 0.0069 - mape: 11.1272
Epoch 183/200
2/2 [==============================] - 0s 30ms/step - loss: 0.0069 - mape: 11.1264
Epoch 184/200
2/2 [==============================] - 0s 35ms/step - loss: 0.0069 - mape: 11.1255
Epoch 185/200
2/2 [==============================] - 0s 27ms/step - loss: 0.0069 - mape: 11.1247
Epoch 186/200
2/2 [==============================] - 0s 45ms/step - loss: 0.0069 - mape: 11.1239
Epoch 187/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0069 - mape: 11.1231
Epoch 188/200
2/2 [==============================] - 0s 37ms/step - loss: 0.0069 - mape: 11.1222
Epoch 189/200
2/2 [==============================] - 0s 44ms/step - loss: 0.0069 - mape: 11.1214
Epoch 190/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0069 - mape: 11.1205
Epoch 191/200
2/2 [==============================] - 0s 33ms/step - loss: 0.0069 - mape: 11.1197
Epoch 192/200
2/2 [==============================] - 0s 32ms/step - loss: 0.0069 - mape: 11.1188
Epoch 193/200
2/2 [==============================] - 0s 33ms/step - loss: 0.0069 - mape: 11.1180
Epoch 194/200
2/2 [==============================] - 0s 27ms/step - loss: 0.0069 - mape: 11.1171
Epoch 195/200
2/2 [==============================] - 0s 29ms/step - loss: 0.0069 - mape: 11.1163
Epoch 196/200
2/2 [==============================] - 0s 31ms/step - loss: 0.0069 - mape: 11.1155
Epoch 197/200
2/2 [==============================] - 0s 34ms/step - loss: 0.0069 - mape: 11.1146
Epoch 198/200
2/2 [==============================] - 0s 36ms/step - loss: 0.0069 - mape: 11.1138
Epoch 199/200
2/2 [==============================] - 0s 35ms/step - loss: 0.0069 - mape: 11.1130
Epoch 200/200
2/2 [==============================] - 0s 36ms/step - loss: 0.0069 - mape: 11.1121
```

In [122…

```
# Print model description
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                4352
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 264
_________________________________________________________________
reshape_1 (Reshape)          (None, 8, 1)              0
=================================================================
Total params: 4,616
Trainable params: 4,616
Non-trainable params: 0
_________________________________________________________________
```

## Evaluate Performance

In [123…

```
# Evaluate MAPE for training data
model.evaluate(window.make_dataset(df_train))
```

```
2/2 [==============================] - 1s 14ms/step - loss: 0.0069 - mape: 11.1051
```

Out[123]:

```
[0.006896934937685728, 11.105128288269043]
```

In [124…

```
# Evaluate MAPE for testing data.
```

```
# The MAPE is less than 5%, so the model is acceptable.
# The testing data performs much better than training data in terms of MAPE.
# That is because tthe acclaim feature is scaled from around 40-70 to 0-1.
# The training performance should have been similar in terms of residues.
# However, the training data start from around 40, which is scaled to 0.
# The scaled training data is then sensitive to small changes in term of percentage.
model.evaluate(window.make_dataset(df_test))
```

```
1/1 [==============================] - 0s 381ms/step - loss: 7.7003e-04 - mape: 2.2462
```

Out[124]:
```
[0.0007700334535911679, 2.246246337890625]
```

In [125…
```
# Evaluate model's prediction against actual data during COVID-19 period.
# If MAPE is lower than 5%, we can say that the anime acclaim is unaffected by COVID-19.
# Otherwise we can postulate that it has been affected by COVID-19.
model.evaluate(window.make_dataset(df_predict))
```

```
1/1 [==============================] - 0s 200ms/step - loss: 0.0132 - mape: 13.0901
```

Out[125]:
```
[0.013173935934901237, 13.090118408203125]
```

## Plot Result

In [126…
```
# Predict testing data, reshape testing prediction, and undo the scale
test_pred_dataset = model.predict(window.make_dataset(df_test))
test_pred_array = test_pred_dataset[0].reshape(1, -1)[0]
test_pred_acclaim = scaler.inverse_transform([test_pred_array]).reshape(1, -1)[0]
test_pred_acclaim
```

Out[126]:
```
array([69.88435802, 70.25086251, 70.64691991, 71.00519686, 71.33419385,
       71.6207415 , 71.85922217, 72.08670226])
```

In [127…
```
# Reshape testing data, and undo the scale
test_actual_dataset = df_test["acclaim"]
test_actual_array = test_actual_dataset.to_numpy()[8:]
test_actual_acclaim = scaler.inverse_transform([test_actual_array]).reshape(1, -1)[0]
test_actual_acclaim
```

Out[127]:
```
array([69.72235796, 69.86565855, 70.46891667, 70.68923217, 72.13572656,
       73.21792629, 72.9117912 , 71.64347623])
```

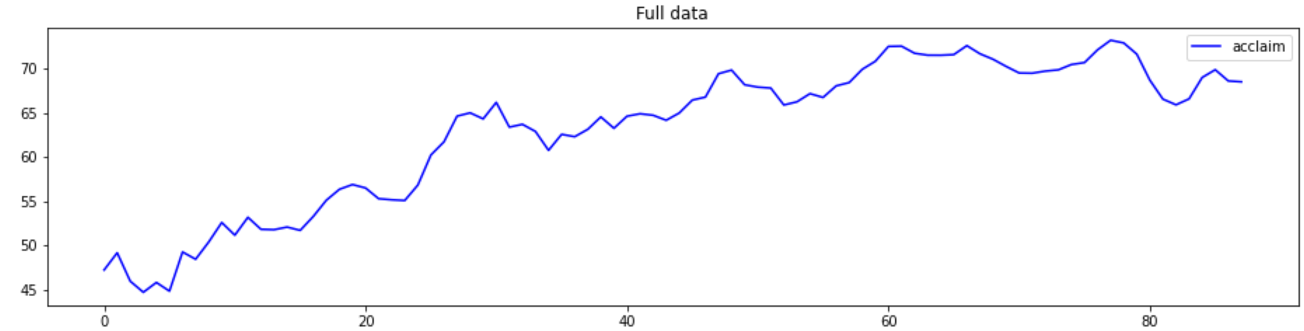
In [128…
```
# Predict prediction data, reshape prediction, and undo the scale
predict_pred_dataset = model.predict(window.make_dataset(df_predict))
predict_pred_array = predict_pred_dataset[0].reshape(1, -1)[0]
predict_pred_acclaim = scaler.inverse_transform([predict_pred_array]).reshape(1, -1)[0]
predict_pred_acclaim
```

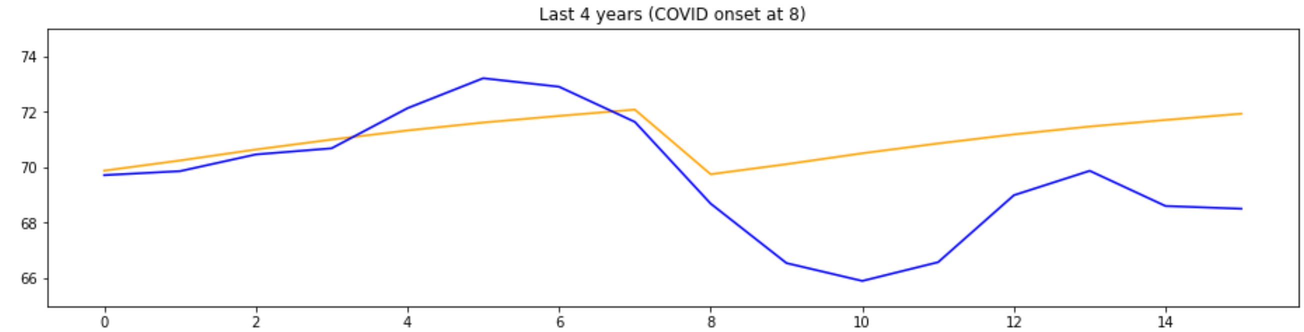
Out[128]:
```
array([69.75529556, 70.11851103, 70.51120618, 70.86631061, 71.19279091,
       71.47710815, 71.71371627, 71.93937042])
```

In [129…
```
# Reshape prediction data, and undo the scale
predict_actual_dataset = df_predict["acclaim"]
predict_actual_array = predict_actual_dataset.to_numpy()[8:]
predict_actual_acclaim = scaler.inverse_transform([predict_actual_array]).reshape(1, -1)[0]
predict_actual_acclaim
```

Out[129]:
```
array([68.69501788, 66.54972141, 65.90647458, 66.58189073, 68.99894725,
       69.87876463, 68.60855258, 68.51046829])
```

In [130…
```
# Concatenate testing and prediction data, and plot the prediction versus actual data.
# The prediction comes in bundle of 8 timesteps.
# Since 8 new timesteps is given for prediction data, the prediction takes a dip due to the new data given.
pred_acclaim = np.concatenate([test_pred_acclaim, predict_pred_acclaim])
actual_acclaim = np.concatenate([test_actual_acclaim, predict_actual_acclaim])

fig, axes = plt.subplots(2, figsize=(16, 8))

df_model.plot(ax=axes[0], color="blue", title="Full data")

plt.plot(pred_acclaim, color="orange")
plt.plot(actual_acclaim, color="blue")
axes[1].set_ylim([65, 75])
axes[1].set_title("Last 4 years (COVID onset at 8)")

plt.show()
```




From model performance metric, the model is acceptable.

Yet, it does not predict a dip in acclaim after the onset of COVID-19.

Therefore, we can say that the COVID-19 pandemic is correlated to the decrase in anime popularity.

One of the reasons why this could happen is due to the loss in production values.

Since the anime sales fell, we may expect the quality to also fall.

This could result in loss of rating and thus, loss of acclaim.

Another reason is that the new animes have not gained enough audience.

Even though the COVID-19 pandemic could have allowed more people watch more anime, the new seasonal animes have not obtained as much viewership as the old classics.

However, this may not be as likely since there are many famous animes like Jujutsu Kaisen and Attack on Titan that garnered over a million of audience.

It is also possible that COVID-19 causes a reduction to TV slots for new series to be aired, effectively letting old shows be rerun, reducing the cost.

In any case, both ARIMA and LSTM models show that the dip in anime acclaim is caused by unwarranted fluctuations, most possibly by COVID-19.

### So, my conclusion is that the loss of sales since 2020 due to COVID-19 have perhaps affected the anime quality, but definitely affected the rating and viewership of animes in general.

# References


MyAnimeList: https://myanimelist.net/anime/season/archive

Web Scraper: https://webscraper.io/

ARIMA model reference: https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

LSTM model reference: https://www.tensorflow.org/tutorials/structured_data/time_series

Anime sales drop: https://www.japantimes.co.jp/news/2021/08/15/business/anime-industry-sales/