Photo Credit:
The Vintage Watch Company / Pavilion

Introduction

Since the onset of Covid, online shopping grew rapidly. Thanks to the numerous online market places, we can easily track the price of certain merchandises and uses data science models to capture the pricing structure.

In this project we use data scrapped from Chrono24.com, one of the most popular used luxury watch listing websites, to predict Rolex watches's prices.

Data Wrangling

A quick scan of the dataset:

	index	model	referen...	price	additiona...	ad name	movement	*case material*	case diameter	year of ...	condition	scope of delivery	location
42040	42040	Rolex Date...	16013	5830	0	diamond 3...	Automatic	Aluminum	NaN	2000	Very good	No original box, no original ...	United States of Americ...
3541	3541	Rolex	Rolex Clas...	1321	126	Tonneau 1...	Manual winding	Bronze	28 mm	1941	Good	No original box, no original ...	Gibraltar, Gibraltar
5717	5717	Rolex	1178	972	0	Rolco Role...	Manual winding	Bronze	31 x 33 mm	1920	Poor	No original box, no original ...	Italy, Precenicco
23379	23379	Rolex	NaN	15770	53	455 Time t...	Quartz	Bronze	NaN	1980	Very good	No original box, no original ...	Germany, München
40589	40589	Rolex	NaN	15770	53	455 Time t...	Quartz	Bronze	NaN	1980	Very good	No original box, no original ...	Germany, München
8732	8732	Rolex Dayt...	116506	nan	300	Skeleton C...	Automatic	Carbon	40 mm	nan	New	No original box, no original ...	United States of Americ...
647	647	Rolex Dayt...	116515 LN	70307	53	Cosmogra...	Automatic	Ceramic	40 mm	nan	New	Original box, original papers	Germany, Bamberg
664	664	Rolex Dayt...	116518LN	42850	0	Cosmogra...	Automatic	Ceramic	40 mm	nan	Unworn	Original box, original papers	United States of Americ...
4367	4367	Rolex Dayt...	116518LN	66943	0	116518LN ...	Automatic	Ceramic	40.00 mm	nan	New	Original box, original papers	Denmark, Copenhagen
7714	7714	Rolex Dayt...	116515ln	nan	1000	116515ln	Automatic	Ceramic	40 mm	2022	New	Original box, original papers	UAE, Dubai
7837	7837	Rolex Dayt...	rolex dayt...	25000	99	rolex dayt...	Automatic	Ceramic	NaN	2003	Very good	No original box, no original ...	United States of Americ...
10179	10179	Rolex Dayt...	116518LN	59000	100	116518LN	Automatic	Ceramic	40 mm	2019	Very good	Original box, original papers	United States of Americ...
12298	12298	Rolex Sub...	116613LB	18200	0	116613LB	Automatic	Ceramic	41 mm	2014	Very good	Original box, original papers	United States of Americ...
12857	12857	Rolex Dayt...	116515ln	49000	200	Chocolate ...	Automatic	Ceramic	40 mm	2022	New	Original box, original papers	United States of Americ...
14305	14305	Rolex Sea-...	126600	14790	150	Red Cera...	Automatic	Ceramic	43 mm	2019	Very good	Original box, no original pa...	United States of Americ...
15451	15451	Rolex Sea-...	126600	15295	99	Red Sea D...	Automatic	Ceramic	43 mm	2019	Very good	Original box, original papers	United States of Americ...

The dataset has 87117 entries in total

```
df.shape
✓ 0.0s
(87117, 12)
```

We need to have an idea of the different data types in each column before proceeding to clean the data.

```
df.dtypes
```

```
model                          object
reference number               object
price                         float64
aditional shipping price      float64
ad name                        object
movement                       object
case material                  object
case diameter                  object
year of production            float64
condition                      object
scope of delivery              object
location                       object
dtype: object
```

Since the price will be our target variable, we should drop the entries that don't have a price.

```python
#check nan in price and drop them, since the price will be our target variable
df['price'].isna().sum()
```

Python

```
5392
```

```python
df = df.dropna(subset=['price'])
df['price'].isna().sum()
```

Python

```
0
```

Now let's deal with missing values in 'case diameter'
First, lets take a look at what the entries that lack 'case diameter' information look like

```python
#check NAs in case diamter
df['case diameter'].isna().sum()
```

```
3840
```

```python
print(df[df['case diameter'].isna()].sample(20))
```

```
                              model reference number     price  \
37651  Rolex Oyster Perpetual Lady Date            79240    3027.0
45130                    Rolex Daytona           116505   45835.0
69067            Rolex Yacht-Master 40           126621   20599.0
30729                   Rolex Day-Date              NaN   19500.0
64202                Rolex Day-Date 36            18238   17407.0
42193                    Rolex Daytona           116528   61285.0
58844        Rolex Oyster Perpetual 31            67480    3531.0
2087                    Rolex Explorer           114270    6172.0
33847            Rolex Yacht-Master 40           126622   15890.0
5746               Rolex GMT-Master II       126720VTNR   34490.0
22532      Rolex Oyster Perpetual Date            15223    6437.0
68143                    Rolex Oyster             4444    4105.0
71119           Rolex Oyster Perpetual            76198    6967.0
75302                    Rolex Daytona           16528  162022.0
9929             Rolex Yacht-Master 40           16628   29767.0
31448              Rolex Lady-Datejust           179160    5098.0
45856              Rolex Lady-Datejust            69178    7500.0
73979                Rolex Day-Date 36           118238   37080.0
21427              Rolex Lady-Datejust            79174    6643.0
56266       Rolex Submariner (No Date)             5513   20804.0

       aditional shipping price  \
37651                       0.0
45130                     175.0
...
45856  United States of America, California, Beverly ...
73979            United States of America, Texas, Dallas
21427            United States of America, Texas, Dallas
56266                                  Hong Kong, Central
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

The strategy to deal with missing case diameter is this:
first convert the existing case diameter from string to simple numeric data, then fill the missing values with the mode of the case diameter values (the most common case diameter) of that given model

```
df['case diameter'] = df['case diameter'].str[:2]
```

```
print(df['case diameter'].value_counts())
```

```
case diameter
40    27517
36    15447
41    10585
26     5052
31     4238
34     4036
42     3587
44     1521
39     1091
28      940
43      639
37      535
29      461
35      458
25      304
32      272
24      233
30      217
38      170
33      168
15       46
23       42
48       35
16       35
...
52        1
8         1
4         1
Name: count, dtype: int64
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
df['case diameter'] = pd.to_numeric(df['case diameter'], errors='coerce', downcast='integer')
```

Filling the missing case diameter values.

```
# fill NAs in case diamter with the most frequent case diameter in that model
df['case diameter'] = df.groupby('model')['case diameter'].transform(lambda x: x.fillna(x.mode().iloc[0]))
```

```
df['case diameter'].isna().sum()
```

```
0
```

Next with deal with the model column. Since we already have the case diameter information in another column, we can delete the unnecessary case diameter information in the model name column. So if there is a two digit number at the end of the model names, we get rid of them.

```python
#remove unnecessary information from the model name -- case diameter
df['model'] = df['model'].str.replace(r' \d{2}$', '', regex=True)
```

```python
df['model'].value_counts()
```

```
model
Rolex Datejust                     20471
Rolex Daytona                       9143
Rolex Submariner Date               6556
Rolex GMT-Master II                 5351
Rolex Lady-Datejust                 5113
Rolex Day-Date                      4947
Rolex Oyster Perpetual              4848
Rolex Yacht-Master                  2758
Rolex Sky-Dweller                   2467
Rolex Explorer II                   2041
Rolex Submariner (No Date)          1735
Rolex Oyster Perpetual Date         1712
Rolex Air King                      1658
Rolex                               1553
Rolex Explorer                      1487
Rolex Milgauss                      1065
Rolex GMT-Master                     947
Rolex Sea-Dweller Deepsea            936
Rolex Datejust II                    816
Rolex Sea-Dweller                    808
Rolex Oyster Perpetual Lady Date     726
Rolex Yacht-Master II                619
Rolex Sea-Dweller 4000               587
Rolex Cellini                        503
...
Rolex Cellini Prince                  38
Rolex Cellini Moonphase               34
Rolex Cellini Date                    33
Rolex Air King Date                   20
Name: count, dtype: int64
```

Lets fill the missing values in movement and case material in the same way as the case diameter. Simply group the entries by model and fill the NAs with the mode of the value:

```python
df.movement.isna().sum()
```

Python

```
2534
```

```python
# fill missing movement with the most common type in that model
df['movement'] = df['movement'].fillna(df.groupby('model')['movement'].transform(lambda x: x.mode()[0]))
```

Python

```
df['case material'].isna().sum()
```

Python

```
4490
```

```
#fill missing case material with the most common type in that model
df['case material'] = df['case material'].fillna(df.groupby('model')['case material'].transform(lambda x: x.mode()[0]))
```

Python

For the missing values in the year of production, we use the model name and the condition as the group by filter. Then fill the missing year of production as the median value of the filtered data.

```
df['year of production'].isna().sum()
```

Python

```
22154
```

```
roduction with the median value of watches in the same model and condition
df['year of production'].fillna(df.groupby(['model','condition'])['year of production'].transform(lambda x: x.median()))
```

Python

However, there are still some remaining missing values in the year of production, this is probably because there are no other entries sharing the same model name and condition. In this case, we use the reference number instead.

```
# the remaining missing year of production, fill them with the mode of the same reference number
df['year of production'] = df['year of production'].fillna(df.groupby(['reference number'])['year of production'].transfor
```

Python

```
df['year of production'].isna().sum()
```

Python

```
0
```

Now we have gotten rid of all the missing values in year of production. We also checked that the “condition” and “scope of delivery” column don’t have any missing data.
Lastly, we simplify the “location” data, so that only the country data is included.

```
df['location'].value_counts()
```

```
location
United States of America, New York, New York          5986
Japan, Tokyo                                          3900
United Kingdom, London                                3120
United States of America, Florida, Miami              2548
United States of America, Texas, Dallas               2222
                                                      ...
The Netherlands, Wateringen                              1
Saudi Arabia, Saudi Arabia                               1
France, AJACCIO                                          1
United States of America, Massachusetts, Wakefield       1
United Kingdom, Littlewick Green                         1
Name: count, Length: 4079, dtype: int64
```

```
df['location'].isna().sum()
```

```
0
```

```
# save only the country part of the location data
df['country'] = df['location'].str.split(',').str[0]
df['country'].value_counts()
```

```
country
United States of America    28715
Japan                       11155
Italy                        8199
Germany                      7641
United Kingdom               6707
                            ...
Panama                          1
Iceland                         1
Kazakhstan                      1
Iraq                            1
Cambodia                        1
Name: count, Length: 87, dtype: int64
```

This is what the cleaned dataset looks like:

/Users/deanchen/Documents/DS/Capstone 2/Data_Wrangling_2.ipynb > new_df (81725, 9)

	index	model	price	movem...	case m...	case di...	year of ...	condition	scope o...	country
0	0	Rolex Lady...	9080	Automatic	Steel	26	2014	Very good	Original bo...	United Sta...
1	1	Rolex Chro...	16202	Manual wi...	Steel	33	1934	Very good	Original bo...	Italy
2	2	Rolex Dayt...	41567	Automatic	White gold	39	2005	Fair	Original pa...	Japan
3	3	Rolex Sub...	19795	Automatic	Steel	40	2020	Unworn	Original bo...	United Sta...
4	4	Rolex Sub...	10674	Automatic	Steel	40	1990	Good	Original bo...	Japan
5	5	Rolex Date...	20935	Automatic	Gold/Steel	41	2022	New	Original bo...	Hong Kong
6	6	Rolex	6038	Automatic	Steel	36	1999	Very good	No original...	Germany
7	7	Rolex Lady...	7843	Automatic	Steel	26	2003	Very good	Original bo...	United Sta...
8	8	Rolex Date...	5887	Automatic	Steel	36	1978	Very good	No original...	The Nethe...
9	9	Rolex Date...	5725	Automatic	Gold/Steel	36	1977	Good	No original...	Italy
10	10	Rolex Sub...	40774	Automatic	White gold	40	2008	Very good	No original...	The Nethe...
11	11	Rolex Day-...	11342	Automatic	Yellow gold	36	1957	Very good	No original...	Italy
12	12	Rolex Day-...	46650	Automatic	Rose gold	40	2022	New	Original bo...	Canada
13	13	Rolex Date...	4591	Automatic	Steel	36	1989	Good	No original...	Italy
14	15	Rolex Dayt...	25922	Automatic	Steel	40	2013	Very good	Original pa...	Italy
15	16	Rolex Date...	14950	Automatic	Yellow gold	36	2005	Very good	Original bo...	United Sta...

EDA

We take a look at all the numeric values to see if there are obvious outliers:

```
df.groupby('model').describe()
```

✓ 0.2s

	price								case diameter		...			year of production						
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%	max
model																					
Rolex	1553.0	12734.292337	25683.637425	315.0	3299.00	5671.0	13995.00	340000.0	1553.0	31.591114	...	35.0	52.0	1553.0	1971.394720	25.547088	1900.0	1954.000	1972.0	1980.00	2022.0
Rolex Air King	1658.0	7671.613993	3391.042297	323.0	4806.00	6447.0	10379.00	30420.0	1658.0	36.371532	...	40.0	49.0	1658.0	2003.911641	16.308642	1940.0	1997.000	2002.0	2019.00	2022.0
Rolex Air King Date	20.0	4649.700000	666.525800	3356.0	4259.00	4838.5	5073.25	5721.0	20.0	34.000000	...	34.0	34.0	20.0	1971.275000	6.540310	1960.0	1966.875	1971.0	1972.00	1988.0
Rolex Cellini	503.0	6221.341948	5784.341606	1498.0	3913.00	5066.0	6999.50	104774.0	503.0	30.566600	...	32.0	47.0	503.0	1985.974155	21.081428	1600.0	1976.000	1990.0	1993.00	2022.0
Rolex Cellini Danaos	41.0	6365.634146	2170.362501	3134.0	4767.00	5549.0	7493.00	10791.0	41.0	35.195122	...	38.0	38.0	41.0	2006.414634	5.253454	1999.0	2003.000	2005.0	2007.00	2022.0
Rolex Cellini Date	33.0	16685.666667	2573.762028	10736.0	15850.00	16292.0	17299.00	22430.0	33.0	39.000000	...	39.0	39.0	33.0	2017.181818	1.530003	2015.0	2016.000	2017.0	2017.00	2021.0
Rolex Cellini Dual Time	42.0	16550.261905	2129.661442	12962.0	15148.75	15950.0	18102.50	23075.0	42.0	39.000000	...	39.0	39.0	42.0	2018.166667	2.722236	2013.0	2017.250	2018.0	2021.00	2022.0
Rolex Cellini Moonphase	34.0	29490.558824	3664.027129	23993.0	27004.00	28587.0	31500.00	37926.0	34.0	39.000000	...	39.0	39.0	34.0	2019.470588	1.910661	2013.0	2019.000	2019.0	2021.00	2022.0
Rolex Cellini Prince	38.0	13465.631579	2495.833928	5000.0	12028.25	13080.5	15033.75	18308.0	38.0	29.605263	...	28.0	47.0	38.0	2002.078947	21.728524	1930.0	2007.000	2007.0	2007.00	2021.0
Rolex Cellini Time	55.0	15549.327273	4137.766024	3516.0	13052.50	14747.0	17272.00	24735.0	55.0	38.763636	...	39.0	39.0	55.0	2016.327273	6.143514	1976.0	2015.000	2016.0	2019.00	2022.0
Rolex Chronograph	61.0	48757.426230	31994.681523	3000.0	25383.00	44981.0	62648.00	161914.0	61.0	35.131148	...	36.0	37.0	61.0	1947.229508	12.380621	1930.0	1938.000	1945.0	1961.00	1969.0
Rolex Datejust	20471.0	12669.601680	6431.272567	695.0	7982.00	11995.0	15973.50	149000.0	20471.0	36.568756	...	41.0	48.0	20471.0	2988.907332	140134.391436	1884.0	2005.000	2018.0	2022.00	20052004.0
Rolex Datejust II	816.0	12801.757353	2908.541507	7842.0	10828.75	12422.0	13995.00	29999.0	816.0	40.966912	...	41.0	49.0	816.0	2014.455882	2.701604	1995.0	2013.750	2014.0	2016.00	2022.0
Rolex Datejust Oysterquartz	244.0	9509.311475	14138.849303	3819.0	5941.00	6956.0	8258.50	135018.0	244.0	36.098361	...	36.0	39.0	244.0	1983.702869	5.499984	1970.0	1980.000	1983.0	1987.00	2000.0
Rolex Datejust Turn-O-Graph	497.0	8280.303823	2819.666572	3000.0	6715.00	7790.0	9400.00	34565.0	497.0	36.014085	...	36.0	40.0	497.0	1996.853119	13.252075	1948.0	1991.000	2003.0	2004.00	2019.0
Rolex Day-Date	4947.0	48573.564382	31475.598118	795.0	24303.00	46085.0	61644.00	377431.0	4947.0	37.709723	...	40.0	45.0	4947.0	2006.977158	16.773656	1955.0	1996.000	2017.0	2022.00	2022.0
Rolex Day-Date II	340.0	62864.414706	21054.731695	28620.0	48987.50	59999.5	71275.00	189025.0	340.0	40.985294	...	41.0	41.0	340.0	2013.088235	2.578189	2008.0	2012.000	2013.0	2015.00	2022.0
Rolex Day-Date Oysterquartz	84.0	22510.738095	7761.260825	11993.0	16500.00	19335.0	26681.50	43152.0	84.0	36.000000	...	36.0	36.0	84.0	1982.928571	4.068053	1978.0	1980.000	1981.5	1986.00	1997.0
Rolex Daytona	9143.0	57225.727989	58936.085445	5000.0	31864.00	44400.0	60000.00	1152102.0	9143.0	39.941376	...	40.0	50.0	9143.0	2012.439407	10.195834	1962.0	2006.000	2014.0	2022.00	2022.0
Rolex Explorer	1487.0	10671.577673	6677.041635	2441.0	7564.00	9275.0	11260.00	85551.0	1487.0	36.682582	...	36.0	44.0	1487.0	2006.838601	14.189955	1946.0	2002.000	2007.0	2019.00	2022.0
Rolex Explorer II	2041.0	12559.859873	5349.058447	6799.0	9721.00	11500.0	13502.00	75000.0	2041.0	40.903479	...	42.0	54.0	2041.0	2007.339539	11.110358	1970.0	2000.000	2005.0	2018.00	2022.0
Rolex GMT-Master	947.0	23883.747624	19741.428922	995.0	14202.50	17822.0	25995.50	199827.0	947.0	39.912355	...	40.0	40.0	947.0	1981.139916	11.399562	1954.0	1972.000	1982.0	1989.00	2022.0
Rolex GMT-Master II	5351.0	27966.240142	26040.580086	8000.0	17499.50	21299.0	26202.50	340497.0	5351.0	40.026163	...	40.0	51.0	5351.0	2014.934218	8.520924	1983.0	2011.000	2018.0	2022.00	2023.0
Rolex Lady-Datejust	5113.0	8872.777821	6876.591482	2475.0	4969.00	6753.0	10547.00	120822.0	5113.0	26.421670	...	26.0	36.0	5113.0	1999.137493	11.437422	1948.0	1991.000	1997.0	2003.00	2022.0
Rolex Lady-Datejust Pearlmaster	290.0	24192.493103	12187.667083	9100.0	16677.50	20497.0	26869.25	78000.0	290.0	29.410345	...	29.0	39.0	290.0	2003.420690	6.893825	1991.0	2000.000	2001.0	2007.00	2022.0
Rolex Milgauss	1065.0	13319.837559	3929.735454	7347.0	11350.00	12835.0	14462.00	54007.0	1065.0	39.954930	...	40.0	49.0	1065.0	2014.464789	7.672296	1965.0	2010.000	2014.0	2021.00	2023.0
Rolex Oyster	165.0	3804.921212	2355.674648	999.0	2700.00	3267.0	4159.00	19108.0	165.0	32.660606	...	34.0	41.0	165.0	1953.345455	10.572574	1930.0	1946.000	1952.0	1959.00	1977.0

It looks like both the price and the year of production has some anomalies. Lets deal with the price first.

```
df.groupby('model')['price'].describe()
```

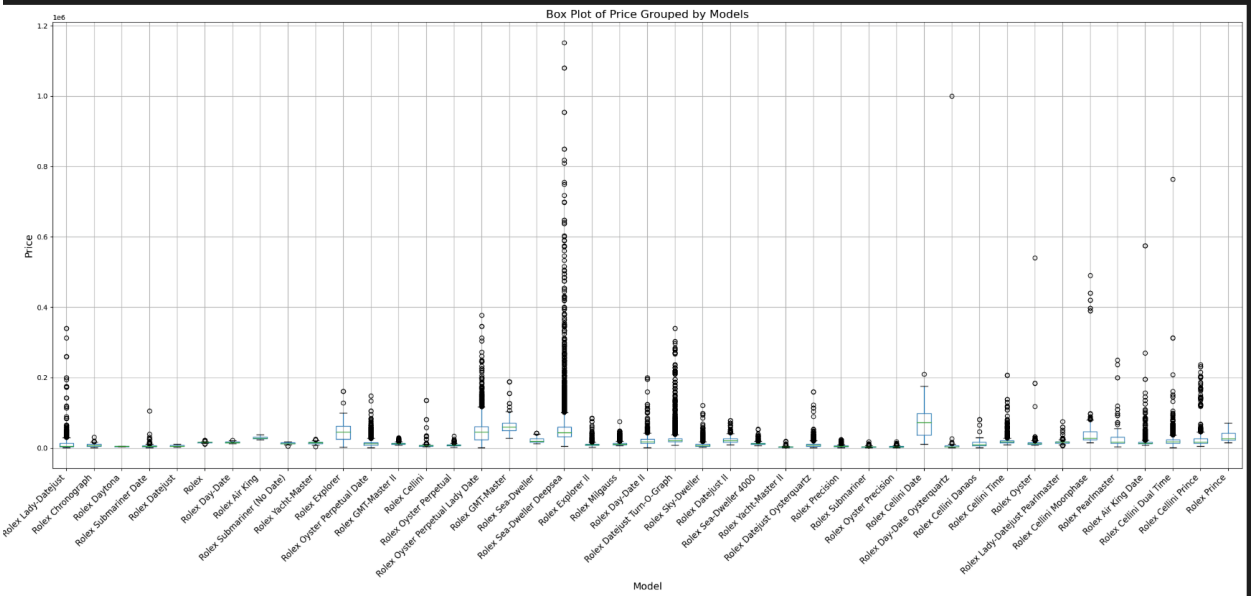
✓ 0.0s

	count	mean	std	min	25%	50%	75%	max
model								
Rolex	1553.0	12734.292337	25683.637425	315.0	3299.00	5671.0	13995.00	340000.0
Rolex Air King	1658.0	7671.613993	3391.042297	323.0	4806.00	6447.0	10379.00	30420.0
Rolex Air King Date	20.0	4649.700000	666.525800	3356.0	4259.00	4838.5	5073.25	5721.0
Rolex Cellini	503.0	6221.341948	5784.341606	1498.0	3913.00	5066.0	6999.50	104774.0
Rolex Cellini Danaos	41.0	6365.634146	2170.362501	3134.0	4767.00	5549.0	7493.00	10791.0
Rolex Cellini Date	33.0	16685.666667	2573.762028	10736.0	15850.00	16292.0	17299.00	22430.0
Rolex Cellini Dual Time	42.0	16550.261905	2129.661442	12962.0	15148.75	15950.0	18102.50	23075.0
Rolex Cellini Moonphase	34.0	29490.558824	3664.027129	23993.0	27004.00	28587.0	31500.00	37926.0
Rolex Cellini Prince	38.0	13465.631579	2495.833928	5000.0	12028.25	13080.5	15033.75	18308.0
Rolex Cellini Time	55.0	15549.327273	4137.766024	3516.0	13052.50	14747.0	17272.00	24735.0
Rolex Chronograph	61.0	48757.426230	31994.681523	3000.0	25383.00	44981.0	62648.00	161914.0
Rolex Datejust	20471.0	12669.601680	6431.272567	695.0	7982.00	11995.0	15973.50	149000.0
Rolex Datejust II	816.0	12801.757353	2908.541507	7842.0	10828.75	12422.0	13995.00	29999.0
Rolex Datejust Oysterquartz	244.0	9509.311475	14138.849303	3819.0	5941.00	6956.0	8258.50	135018.0
Rolex Datejust Turn-O-Graph	497.0	8280.303823	2819.666572	3000.0	6715.00	7790.0	9400.00	34565.0
Rolex Day-Date	4947.0	48573.564382	31475.598118	795.0	24303.00	46085.0	61644.00	377431.0
Rolex Day-Date II	340.0	62864.414706	21054.731695	28620.0	48987.50	59999.5	71275.00	189025.0
Rolex Day-Date Oysterquartz	84.0	22510.738095	7761.260825	11993.0	16500.00	19335.0	26681.50	43152.0
Rolex Daytona	9143.0	57225.727989	58936.085445	5000.0	31864.00	44400.0	60000.00	1152102.0
Rolex Explorer	1487.0	10671.577673	6677.041635	2441.0	7564.00	9275.0	11260.00	85551.0
Rolex Explorer II	2041.0	12559.859873	5349.058447	6799.0	9721.00	11500.0	13502.00	75000.0
Rolex GMT-Master	947.0	23883.747624	19741.428922	995.0	14202.50	17822.0	25995.50	199827.0
Rolex GMT-Master II	5351.0	27966.240142	26040.580086	8000.0	17499.50	21299.0	26202.50	340497.0

We make a box plot to better visualize the price.

```python
import matplotlib.pyplot as plt


boxplot = df.boxplot(column='price', by='model', figsize=(25,12))

plt.title('Box Plot of Price Grouped by Models', fontsize=16)
plt.suptitle("")  # Remove the default group label
plt.xlabel('Model', fontsize=14)
plt.ylabel('Price', fontsize=14)

# Manually set the x-axis tick positions and labels for more spread-out ticks
x_tick_positions = range(1, len(df['model'].unique()) + 1)  # Adjust tick positions
x_tick_labels = df['model'].unique()  # Use unique values from the 'model' column
plt.xticks(x_tick_positions, x_tick_labels, rotation=45, ha='right', fontsize=12)

plt.subplots_adjust(bottom=0.2)  # Adjust bottom margin for rotated labels

plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```

✓ 1.4s Pyth



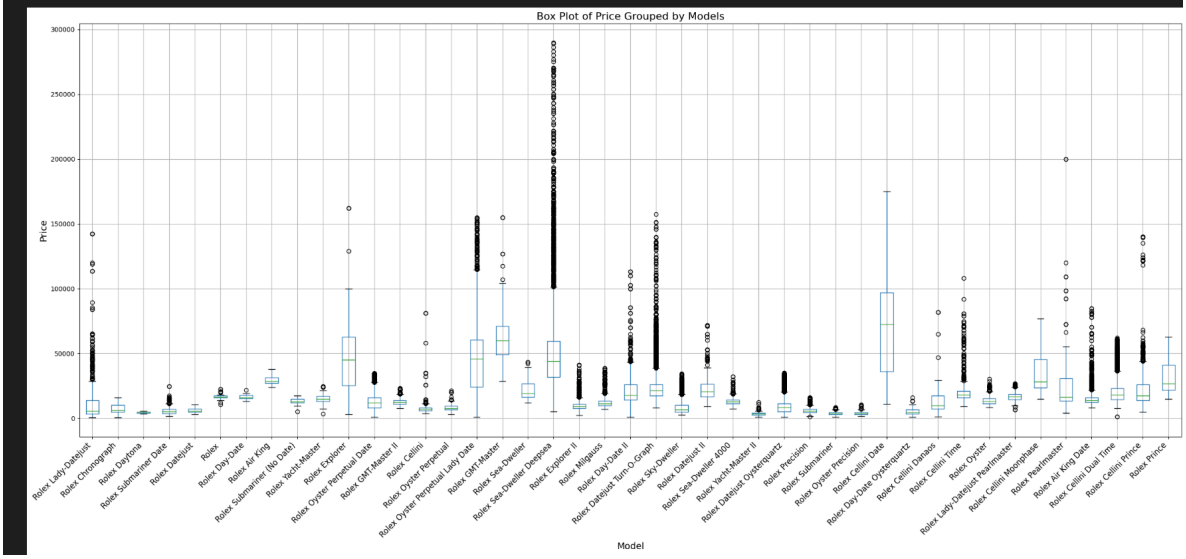
There are plenty of outliers for different models. One way to deal with the outliers is to remove any entries above the 99th percentile.

```python
import numpy as np

def filter_rows(group):
    q = group['price'].quantile(.99)
    return group[group['price'] <= q]

df_filtered = df.groupby('model').apply(filter_rows).reset_index(drop=True)
```

✓ 0.1s Pyt

```python
boxplot = df_filtered.boxplot(column='price', by='model', figsize=(25,12))

plt.title('Box Plot of Price Grouped by Models', fontsize=16)
plt.suptitle("")  # Remove the default group label
plt.xlabel('Model', fontsize=14)
plt.ylabel('Price', fontsize=14)

# Manually set the x-axis tick positions and labels for more spread-out ticks
x_tick_positions = range(1, len(df['model'].unique()) + 1)  # Adjust tick positions
x_tick_labels = df['model'].unique()  # Use unique values from the 'model' column
plt.xticks(x_tick_positions, x_tick_labels, rotation=45, ha='right', fontsize=12)

plt.subplots_adjust(bottom=0.2)  # Adjust bottom margin for rotated labels

plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```

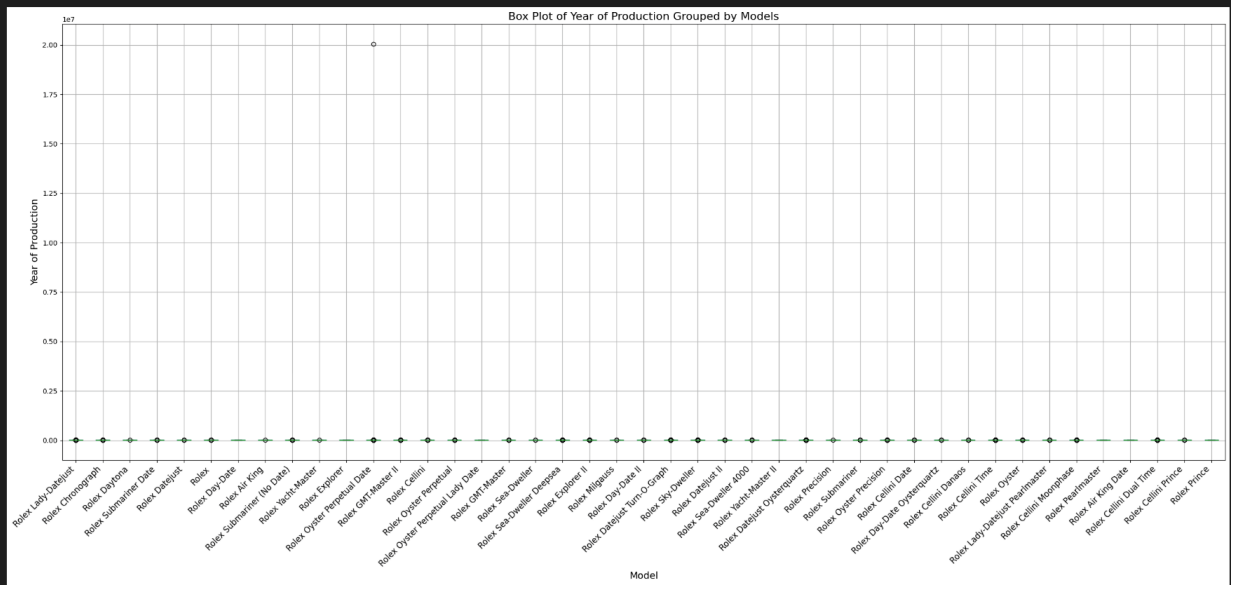
✓ 1.0s Pyt



We see from the new box plot. That strategy was very effective. Now we look at the box plot for the year of production.

```
boxplot = df_filtered.boxplot(column='year of production', by='model', figsize=(25,12))

plt.title('Box Plot of Year of Production Grouped by Models', fontsize=16)
plt.suptitle("")  # Remove the default group label
plt.xlabel('Model', fontsize=14)
plt.ylabel('Year of Production', fontsize=14)

# Manually set the x-axis tick positions and labels for more spread-out ticks
x_tick_positions = range(1, len(df['model'].unique()) + 1)  # Adjust tick positions
x_tick_labels = df['model'].unique()  # Use unique values from the 'model' column
plt.xticks(x_tick_positions, x_tick_labels, rotation=45, ha='right', fontsize=12)

plt.subplots_adjust(bottom=0.2)  # Adjust bottom margin for rotated labels

plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```

✓ 1.0s Pyt



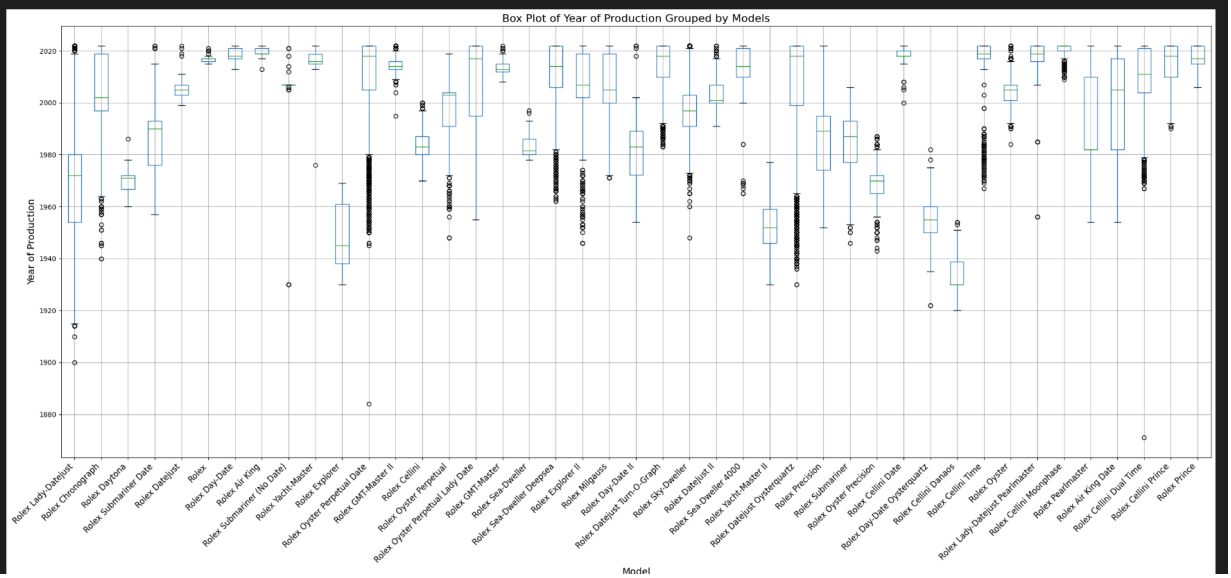
There is one visible outlier. Let’s set the year of production to between 2023 and 1800.

```
df_filtered  = df_filtered[df_filtered['year of production'] < 2023.0]
df_filtered  = df_filtered[df_filtered['year of production'] > 1800.0]
```

✓ 0.0s Pyth

```
boxplot = df_filtered.boxplot(column='year of production', by='model', figsize=(25,12))

plt.title('Box Plot of Year of Production Grouped by Models', fontsize=16)
plt.suptitle("")  # Remove the default group label
plt.xlabel('Model', fontsize=14)
plt.ylabel('Year of Production', fontsize=14)

# Manually set the x-axis tick positions and labels for more spread-out ticks
x_tick_positions = range(1, len(df['model'].unique()) + 1)  # Adjust tick positions
x_tick_labels = df['model'].unique()  # Use unique values from the 'model' column
plt.xticks(x_tick_positions, x_tick_labels, rotation=45, ha='right', fontsize=12)

plt.subplots_adjust(bottom=0.2)  # Adjust bottom margin for rotated labels

plt.tight_layout()  # Adjust layout for better spacing
plt.show()
```

✓ 0.9s Pyth



The box plot looks much better now!
We lastly one hot encode all the categorical data before modeling

```
import category_encoders as ce
encoder = ce.OneHotEncoder(use_cat_names= True)
df_encoded = encoder.fit_transform(df_filtered)
```

✓ 1.6s

```
print(df_encoded.head(10))
```

✓ 0.0s

```
   model_Rolex  model_Rolex Air King  model_Rolex Air King Date  \
0            1                     0                          0
1            1                     0                          0
2            1                     0                          0
3            1                     0                          0
4            1                     0                          0
5            1                     0                          0
6            1                     0                          0
7            1                     0                          0
8            1                     0                          0
9            1                     0                          0

   model_Rolex Cellini  model_Rolex Cellini Danaos  model_Rolex Cellini Date  \
0                    0                           0                         0
1                    0                           0                         0
2                    0                           0                         0
3                    0                           0                         0
4                    0                           0                         0
5                    0                           0                         0
6                    0                           0                         0
7                    0                           0                         0
8                    0                           0                         0
9                    0                           0                         0

   model_Rolex Cellini Dual Time  model_Rolex Cellini Moonphase  \
...
8                              0                             0
9                              0                             0

[10 rows x 159 columns]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Modeling

We first train test split the data:

```
import pandas as pd
df = pd.read_csv('encoded')
```

✓ 1.0s

```
X = df.drop(columns= 'price')
y = df['price']
```

✓ 0.0s

It's time to split the data into training and test sets!

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

✓ 0.7s

Model 1 Linear Regression

```
from sklearn.linear_model import LinearRegression
reg  = LinearRegression()
reg.fit(X_train,y_train)
```

✓ 0.6s

```
▾ LinearRegression
LinearRegression()
```

```
y_pred  = reg.predict(X_test)
```

✓ 0.0s

```
from sklearn.metrics import mean_squared_error, r2_score
mse  = mean_squared_error(y_test, y_pred)
r_squared  = r2_score(y_test, y_pred)
```

✓ 0.0s

```
print('Mean_Squared_Error :' ,mse)       #Least_MSE
print('r_square_value :',r_squared)      #Highest R_squared
```

✓ 0.0s

```
Mean_Squared_Error : 8.558254896756142e+25
r_square_value : -1.8123189869958102e+17
```

The linear regression model yields a negative R^2 score, which means that linear regression is not a good way of modeling the data.
We try **model 2, random forest regressor**

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train,y_train)
y_pred_rf = rf.predict(X_test)
mse_rf  = mean_squared_error(y_test, y_pred_rf)
r_squared_rf  = r2_score(y_test, y_pred_rf)
print('Mean_Squared_Error :' ,mse_rf)       #Least_MSE
print('r_square_value :',r_squared_rf)      #Highest R_squared
```

✓ 32.8s

```
Mean_Squared_Error : 108817168.34490842
r_square_value : 0.7695658487837905
```

This time the R^2 score is .77. This score indicates the random forest regressor modeling is a much more appropriate way of modeling the data.