Sankaran S - 07/07/2025

# Gold price prediction in India

## Importing libraries

In [93]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import yfinance as yf
from pandas import DataFrame
import requests
import re
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
import pickle
import gradio as gr
```

## Data collections

### Extracting USD_INR exchange rate

In [2]:
```
# Collect USD_INR exchange rate using yfinance
usd_inr: DataFrame = yf.download(tickers="USDINR=X", start="2024-01-01", end="2025-07-07") # type: ignore
display(usd_inr.head())
```

```
C:\Users\sanka\AppData\Local\Temp\ipykernel_12996\3593658626.py:2: FutureWarning: YF.download() has changed argument auto_adjust default to
True
  usd_inr: DataFrame = yf.download(tickers="USDINR=X", start="2024-01-01", end="2025-07-07") # type: ignore
[*********************100%***********************]  1 of 1 completed
```

| Price | Close | High | Low | Open | Volume |
|---|---|---|---|---|---|
| Ticker | USDINR=X | USDINR=X | USDINR=X | USDINR=X | USDINR=X |
| Date | | | | | |
| **2024-01-01** | 83.248596 | 83.237999 | 83.150002 | 83.248596 | 0 |
| **2024-01-02** | 83.202599 | 83.343002 | 83.169800 | 83.202599 | 0 |
| **2024-01-03** | 83.257004 | 83.333702 | 83.246201 | 83.257004 | 0 |
| **2024-01-04** | 83.318100 | 83.360298 | 83.202103 | 83.318100 | 0 |
| **2024-01-05** | 83.240601 | 83.271599 | 83.035004 | 83.240601 | 0 |

In [3]:
```
usd_inr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 392 entries, 2024-01-01 to 2025-07-04
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   (Close, USDINR=X)   392 non-null    float64
 1   (High, USDINR=X)    392 non-null    float64
 2   (Low, USDINR=X)     392 non-null    float64
 3   (Open, USDINR=X)    392 non-null    float64
 4   (Volume, USDINR=X)  392 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 18.4 KB
```

In [4]:
```
usd_inr.reset_index(inplace= True)
```

In [5]:
```
usd_inr.head()
```

Out[5]:

| Price | Date | Close | High | Low | Open | Volume |
|---|---|---|---|---|---|---|
| Ticker | | USDINR=X | USDINR=X | USDINR=X | USDINR=X | USDINR=X |
| **0** | 2024-01-01 | 83.248596 | 83.237999 | 83.150002 | 83.248596 | 0 |
| **1** | 2024-01-02 | 83.202599 | 83.343002 | 83.169800 | 83.202599 | 0 |
| **2** | 2024-01-03 | 83.257004 | 83.333702 | 83.246201 | 83.257004 | 0 |
| **3** | 2024-01-04 | 83.318100 | 83.360298 | 83.202103 | 83.318100 | 0 |
| **4** | 2024-01-05 | 83.240601 | 83.271599 | 83.035004 | 83.240601 | 0 |

In [6]:

```
usd_inr.tail()
```

Out[6]:

| Price | Date | Close | High | Low | Open | Volume |
|---|---|---|---|---|---|---|
| Ticker | | USDINR=X | USDINR=X | USDINR=X | USDINR=X | USDINR=X |
| **387** | 2025-06-30 | 85.455299 | 85.927696 | 85.241997 | 85.455299 | 0 |
| **388** | 2025-07-01 | 85.713997 | 85.741699 | 85.459297 | 85.713997 | 0 |
| **389** | 2025-07-02 | 85.642799 | 85.771004 | 85.510498 | 85.642799 | 0 |
| **390** | 2025-07-03 | 85.694099 | 85.697998 | 85.192703 | 85.694099 | 0 |
| **391** | 2025-07-04 | 85.319099 | 85.553001 | 85.307297 | 85.319099 | 0 |

In [7]:

```
#keep only relevent columns
usd_inr = usd_inr[['Date', 'Close']]
```

In [8]:

```
#Changing colun names
usd_inr.columns = ['Date', 'USD_INR']
```

In [9]:

```
usd_inr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Date     392 non-null    datetime64[ns]
 1   USD_INR  392 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 6.3 KB
```

In [10]:
```
usd_inr
```

Out[10]:

| | Date | USD_INR |
|---|---|---|
| **0** | 2024-01-01 | 83.248596 |
| **1** | 2024-01-02 | 83.202599 |
| **2** | 2024-01-03 | 83.257004 |
| **3** | 2024-01-04 | 83.318100 |
| **4** | 2024-01-05 | 83.240601 |
| **...** | ... | ... |
| **387** | 2025-06-30 | 85.455299 |
| **388** | 2025-07-01 | 85.713997 |
| **389** | 2025-07-02 | 85.642799 |
| **390** | 2025-07-03 | 85.694099 |
| **391** | 2025-07-04 | 85.319099 |

392 rows × 2 columns

# Scraping the gold prices from the website

## Extracting 2024 data

In [12]:
```
url_2024 = "https://www.exchange-rates.org/api/v2/rates/GetChartRates?dataDisplayMode=2&from=XAU&to=INR&countryCode=IN&unit=G&karat=22&group
```

```
response_2024 = requests.get(url_2024)
data_2024 = response_2024.json()
```

In [13]:

```
data_2024
```

Out[13]:

{'ErrorMessage': None,
 'SectionHeader': None,
 'StatisticsHtml': ' This chart shows gold prices in India for 2024. Prices are in Indian Rupees per gram\r\n\r\nfor 22K gold\r\n. ',
 'FAQHtml': '<section class="faq-section wo-title"><h2>What was the average gold price in India in 2024?</h2>\n The average price of gold in 2024 was ₹5,895.44 per gram. \n<h2>What was the highest gold price in India in 2024?</h2>\n The high point per gram of\r\n22K\r\ngold during 2024 was ₹6,907.09 on October 30, 2024. \n<h2>What was the lowest gold price in India in 2024?</h2>\n The low point was ₹4,877.01 per gram of gold on February 13, 2024. \n<h2>Was the price of gold in India up or down in 2024?</h2>\n The price of\r\n22K\r\ngold in INR was\r\nup +30.67%\r\nin 2024. \n</section>',
 'ChartData': '[[D(2024,1,1),R(5066311,3)],[D(2024,1,2),R(5053341,3)],[D(2024,1,3),R(5014551,3)],[D(2024,1,4),R(5012479,3)],[D(2024,1,5),R(5015198,3)],[D(2024,1,8),R(4966310,3)],[D(2024,1,9),R(4972306,3)],[D(2024,1,10),R(4955580,3)],[D(2024,1,11),R(4978278,3)],[D(2024,1,12),R(5005296,3)],[D(2024,1,15),R(5015326,3)],[D(2024,1,16),R(4962556,3)],[D(2024,1,17),R(4925980,3)],[D(2024,1,18),R(4956514,3)],[D(2024,1,19),R(4972076,3)],[D(2024,1,22),R(4950837,3)],[D(2024,1,23),R(4974937,3)],[D(2024,1,24),R(4939310,3)],[D(2024,1,25),R(4952733,3)],[D(2024,1,26),R(4944702,3)],[D(2024,1,29),R(4976772,3)],[D(2024,1,30),R(4987353,3)],[D(2024,1,31),R(4998434,3)],[D(2024,2,1),R(5026316,3)],[D(2024,2,2),R(4988710,3)],[D(2024,2,5),R(4953904,3)],[D(2024,2,6),R(4983216,3)],[D(2024,2,7),R(4980184,3)],[D(2024,2,8),R(4974870,3)],[D(2024,2,9),R(4953073,3)],[D(2024,2,12),R(4938727,3)],[D(2024,2,13),R(4877012,3)],[D(2024,2,14),R(4877947,3)],[D(2024,2,15),R(4902940,3)],[D(2024,2,16),R(4924927,3)],[D(2024,2,19),R(4934986,3)],[D(2024,2,20),R(4944551,3)],[D(2024,2,21),R(4952549,3)],[D(2024,2,22),R(4947502,3)],[D(2024,2,23),R(4972228,3)],[D(2024,2,26),R(4960543,3)],[D(2024,2,27),R(4961670,3)],[D(2024,2,28),R(4973105,3)],[D(2024,2,29),R(4994153,3)],[D(2024,3,1),R(5084776,3)],[D(2024,3,4),R(5167643,3)],[D(2024,3,5),R(5199018,3)],[D(2024,3,6),R(5246300,3)],[D(2024,3,7),R(5265414,3)],[D(2024,3,8),R(5314526,3)],[D(2024,3,11),R(5326462,3)],[D(2024,3,12),R(5270927,3)],[D(2024,3,13),R(5308440,3)],[D(2024,3,14),R(5286581,3)],[D(2024,3,15),R(5267138,3)],[D(2024,3,18),R(5283440,3)],[D(2024,3,19),R(5282424,3)],[D(2024,3,20),R(5409553,3)],[D(2024,3,21),R(5351277,3)],[D(2024,3,22),R(5335320,3)],[D(2024,3,25),R(5341214,3)],[D(2024,3,26),R(5352749,3)],[D(2024,3,27),R(5388866,3)],[D(2024,3,28),R(5482022,3)],[D(2024,3,29),R(5480086,3)],[D(2024,4,1),R(5543846,3)],[D(2024,4,2),R(5621963,3)],[D(2024,4,3),R(5664795,3)],[D(2024,4,4),R(5638518,3)],[D(2024,4,5),R(5719690,3)],[D(2024,4,8),R(5737212,3)],[D(2024,4,9),R(5770855,3)],[D(2024,4,10),R(5739387,3)],[D(2024,4,11),R(5835313,3)],[D(2024,4,12),R(5777346,3)],[D(2024,4,15),R(5874865,3)],[D(2024,4,16),R(5883155,3)],[D(2024,4,17),R(5826203,3)],[D(2024,4,18),R(5861506,3)],[D(2024,4,19),R(5874459,3)],[D(2024,4,22),R(5720099,3)],[D(2024,4,23),R(5700703,3)],[D(2024,4,24),R(5693604,3)],[D(2024,4,25),R(5728700,3)],[D(2024,4,26),R(5746608,3)],[D(2024,4,29),R(5744337,3)],[D(2024,4,30),R(5636614,3)],[D(2024,5,1),R(5704486,3)],[D(2024,5,2),R(5667591,3)],[D(2024,5,3),R(5658165,3)],[D(2024,5,6),R(5724768,3)],[D(2024,5,7),R(5697971,3)],[D(2024,5,8),R(5683480,3)],[D(2024,5,9),R(5768909,3)],[D(2024,5,10),R(5812120,3)],[D(2024,5,13),R(5754391,3)],[D(2024,5,14),R(5799497,3)],[D(2024,5,15),R(5873954,3)],[D(2024,5,16),R(5851907,3)],[D(2024,5,17),R(5926215,3)],[D(2024,5,20),R(5966342,3)],[D(2024,5,21),R(5946246,3)],[D(2024,5,22),R(5842616,3)],[D(2024,5,23),R(5724208,3)],[D(2024,5,24),R(5713844,3)],[D(2024,5,27),R(5765038,3)],[D(2024,5,28),R(5786999,3)],[D(2024,5,29),R(5746377,3)],[D(2024,5,30),R(5756350,3)],[D(2024,5,31),R(5724666,3)],[D(2024,6,3),R(5757161,3)],[D(2024,6,4),R(5740172,3)],[D(2024,6,5),R(5790115,3)],[D(2024,6,6),R(5845441,3)],[D(2024,6,7),R(5646929,3)],[D(2024,6,10),R(5691701,3)],[D(2024,6,11),R(5704721,3)],[D(2024,6,12),R(5714250,3)],[D(2024,6,13),R(5671866,3)],[D(2024,6,14),R(5744793,3)],[D(2024,6,17),R(5710884,3)],[D(2024,6,18),R(5722490,3)],[D(2024,6,19),R(5729078,3)],[D(2024,6,20),R(5816223,3)],[D(2024,6,21),R(5716010,3)],[D(2024,6,24),R(5738986,3)],[D(2024,6,25),R(5707851,3)],[D(2024,6,26),R(5662983,3)],[D(2024,6,27),R(5722507,3)],[D(2024,6,28),R(5716677,3)],[D(2024,7,1),R(5734294,3)],[D(2024,7,2),R(5734940,3)],[D(2024,7,3),R(5799169,3)],[D(2024,7,4),R(5797046,3)],[D(2024,7,5),R(5885016,3)],[D(2024,7,8),R(5803071,3)],[D(2024,7,9),R(5816190,3)],[D(2024,7,10),R(5838104,3)],[D(2024,7,11),R(5937962,3)],[D(2024,7,12),R(5935539,3)],[D(2024,7,15),R(5964956,3)],[D(2024,7,16),R(6074906,3)],[D(2024,7,17),R(6056034,3)],[D(2024,7,18),R(6031031,3)],[D(2024,7,19),R(5923910,3)],[D(2024,7,22),R(5909382,3)],[D(2024,7,23),R(5941786,3)],[D(2024,7,24),R(5916808,3)],[D(2024,7,25),R(5833970,3)],[D(2024,7,26),R(5891033,3)],[D(2024,7,29),R(5883334,3)],[D(2024,7,30),R(5945972,3)],[D(2024,7,31),R(6038855,3)],[D(2024,8,1),R(6037492,3)],[D(2024,8,2),R(6028549,3)],[D(2024,8,5),R(5967658,3)],[D(2024,8,6),R(5910225,3)],[D(2024,8,7),R(5895777,3)],[D(2024,8,8),R(6007130,3)],[D(2024,8,9),R(6015088,3)],[D(2024,8,12),R(6114393,3)],[D(2024,8,13),R(6094201,3)],[D(2024,8,14),R(6056490,3)],[D(2024,8,15),R(6073668,3)],[D(2024,8,16),R(6199190,3)],[D(2024,8,19),R(6182135,3)],[D(2024,8,20),R(6210684,3)],[D(2024,8,21),R(6214243,3)],[D(2024,8,22),R(6147102,3)],[D(2024,8,23),R(6205396,3)],[D(2024,8,26),R(6222575,3)],[D(2024,8,27),R(6246080,3)],[D(2024,8,28),R(6203972,3)],[D(2024,8,29),R(6235300,3)],[D(2024,8,30),R(6188792,3)],[D(2024,9,2),R(6179612,3)],[D(2024,9,3),R(6168303,3)],[D(2024,9,4),R(6172506,3)],[D(2024,9,5),R(6227907,3)],[D(2024,9,6),R(6181471,3)],[D(2024,9,9),R(6201840,3)],[D(2024,9,10),R(6229538,3)],[D(2024,9,11),R(6222460,3)],[D(2024,9,12),R(6329455,3)],[D(202

4,9,13),R(6376027,3)],[D(2024,9,16),R(6384571,3)],[D(2024,9,17),R(6346411,3)],[D(2024,9,18),R(6313776,3)],[D(2024,9,19),R(6376508,3)],[D(20
24,9,20),R(6452240,3)],[D(2024,9,23),R(6470861,3)],[D(2024,9,24),R(6548440,3)],[D(2024,9,25),R(6552829,3)],[D(2024,9,26),R(6587192,3)],[D(2
024,9,27),R(6558880,3)],[D(2024,9,30),R(6507663,3)],[D(2024,10,1),R(6583922,3)],[D(2024,10,2),R(6577147,3)],[D(2024,10,3),R(6574266,3)],[D
(2024,10,4),R(6571318,3)],[D(2024,10,7),R(6544004,3)],[D(2024,10,8),R(6486449,3)],[D(2024,10,9),R(6450233,3)],[D(2024,10,10),R(6502508,3)],
[D(2024,10,11),R(6585626,3)],[D(2024,10,14),R(6563327,3)],[D(2024,10,15),R(6595339,3)],[D(2024,10,16),R(6624626,3)],[D(2024,10,17),R(667143
9,3)],[D(2024,10,18),R(6742651,3)],[D(2024,10,21),R(6741680,3)],[D(2024,10,22),R(6810527,3)],[D(2024,10,23),R(6733870,3)],[D(2024,10,24),R
(6778977,3)],[D(2024,10,25),R(6810710,3)],[D(2024,10,28),R(6795347,3)],[D(2024,10,29),R(6874635,3)],[D(2024,10,30),R(6907085,3)],[D(2024,1
0,31),R(6804446,3)],[D(2024,11,1),R(6781262,3)],[D(2024,11,4),R(6786867,3)],[D(2024,11,5),R(6802299,3)],[D(2024,11,6),R(6612189,3)],[D(202
4,11,7),R(6723379,3)],[D(2024,11,8),R(6677390,3)],[D(2024,11,11),R(6523098,3)],[D(2024,11,12),R(6464050,3)],[D(2024,11,13),R(6422685,3)],[D
(2024,11,14),R(6388946,3)],[D(2024,11,15),R(6377126,3)],[D(2024,11,18),R(6498434,3)],[D(2024,11,19),R(6553318,3)],[D(2024,11,20),R(658777
3,3)],[D(2024,11,21),R(6647497,3)],[D(2024,11,22),R(6756975,3)],[D(2024,11,25),R(6524058,3)],[D(2024,11,26),R(6540548,3)],[D(2024,11,27),R
(6559282,3)],[D(2024,11,28),R(6567616,3)],[D(2024,11,29),R(6603743,3)],[D(2024,12,2),R(6597645,3)],[D(2024,12,3),R(6593098,3)],[D(2024,1
2,4),R(6617263,3)],[D(2024,12,5),R(6571604,3)],[D(2024,12,6),R(6570493,3)],[D(2024,12,9),R(6654031,3)],[D(2024,12,10),R(6741796,3)],[D(202
4,12,11),R(6796887,3)],[D(2024,12,12),R(6702679,3)],[D(2024,12,13),R(6621705,3)],[D(2024,12,16),R(6639211,3)],[D(2024,12,17),R(6624482,3)],
[D(2024,12,18),R(6502496,3)],[D(2024,12,19),R(6508444,3)],[D(2024,12,20),R(6567061,3)],[D(2024,12,23),R(6560640,3)],[D(2024,12,24),R(656838
1,3)],[D(2024,12,25),R(6595727,3)],[D(2024,12,26),R(6650442,3)],[D(2024,12,27),R(6597517,3)],[D(2024,12,30),R(6585391,3)],[D(2024,12,31),R
(6620062,3)]]',
 'ChartDataName': 'Gold Price in Indian Rupees'}

In [ ]:

```
# Extract the dates and prices for 2024
chart_data_2024 = data_2024['ChartData']

pattern = r"D\((\d+),(\d+),(\d+)\),R\((\d+),(\d+)\)"

matches_2024 = re.findall(pattern, chart_data_2024)

parsed_data_2024 = []
for year, month, day, value, decimal in matches_2024:
    date = datetime(int(year), int(month), int(day))
    price = int(value) / (10 ** int(decimal)) # e.g: divide by 1000
    parsed_data_2024.append((date, round(price, 2)))

df_2024 = pd.DataFrame(parsed_data_2024, columns=["Date", "gold_rate"])
```

In [16]:

```
df_2024
```

Out[16]:

| | Date | gold_rate |
|---|---|---|
| **0** | 2024-01-01 | 5066.31 |
| **1** | 2024-01-02 | 5053.34 |
| **2** | 2024-01-03 | 5014.55 |
| **3** | 2024-01-04 | 5012.48 |
| **4** | 2024-01-05 | 5015.20 |
| **...** | ... | ... |
| **257** | 2024-12-25 | 6595.73 |
| **258** | 2024-12-26 | 6650.44 |
| **259** | 2024-12-27 | 6597.52 |
| **260** | 2024-12-30 | 6585.39 |
| **261** | 2024-12-31 | 6620.06 |

262 rows × 2 columns

## Extracting 2025 data

In [17]:

```
url_2025 = "https://www.exchange-rates.org/api/v2/rates/GetChartRates?dataDisplayMode=2&from=XAU&to=INR&countryCode=IN&unit=G&karat=22&group

response_2025 = requests.get(url_2025)
data_2025 = response_2025.json()
```

In [18]:

```
data_2025
```

Out[18]:


```
{'ErrorMessage': None,
 'SectionHeader': None,
 'StatisticsHtml': '<p>This chart shows gold prices in India for 2025. Prices are in Indian Rupees per gram\r\n\r\nfor 22K gold\r\n.</p>',
 'FAQHtml': '<section class="faq-section wo-title"><h2>What was the average gold price in India in 2025?</h2>\n<p>The average price of gold
in 2025 was <span class="currencySymbol">₹</span>7,835.23 per gram.</p>\n<h2>What was the highest gold price in India in 2025?</h2>\n<p>The
high point per gram of\r\n22K\r\ngold during 2025 was <span class="currencySymbol">₹</span>8,714.51 on June 13, 2025.</p>\n<h2>What was the
lowest gold price in India in 2025?</h2>\n<p>The low point was <span class="currencySymbol">₹</span>6,617.80 per gram of gold on January 1,
2025.</p>\n<h2>Was the price of gold in India up or down in 2025?</h2>\n<p>The price of\r\n22K\r\ngold in INR was\r\nup <span class="rate-c
hange rate-green">+26.97%</span>\r\nin 2025.</p>\n</section>',
 'ChartData': '[[D(2025,1,1),R(6617799,3)],[D(2025,1,2),R(6723704,3)],[D(2025,1,3),R(6676081,3)],[D(2025,1,6),R(6653734,3)],[D(2025,1,7),R
(6698446,3)],[D(2025,1,8),R(6741716,3)],[D(2025,1,9),R(6762862,3)],[D(2025,1,10),R(6831727,3)],[D(2025,1,13),R(6808163,3)],[D(2025,1,14),R
(6823053,3)],[D(2025,1,15),R(6868059,3)],[D(2025,1,16),R(6925567,3)],[D(2025,1,17),R(6895195,3)],[D(2025,1,20),R(6885729,3)],[D(2025,1,2
1),R(6989108,3)],[D(2025,1,22),R(7022000,3)],[D(2025,1,23),R(7018869,3)],[D(2025,1,24),R(7040096,3)],[D(2025,1,27),R(6985020,3)],[D(202
5,1,28),R(7055387,3)],[D(2025,1,29),R(7048267,3)],[D(2025,1,30),R(7136503,3)],[D(2025,1,31),R(7150338,3)],[D(2025,2,3),R(7211008,3)],[D(202
5,2,4),R(7291511,3)],[D(2025,2,5),R(7384453,3)],[D(2025,2,6),R(7377777,3)],[D(2025,2,7),R(7402813,3)],[D(2025,2,10),R(7506140,3)],[D(202
5,2,11),R(7419231,3)],[D(2025,2,12),R(7439451,3)],[D(2025,2,13),R(7480742,3)],[D(2025,2,14),R(7363282,3)],[D(2025,2,17),R(7416743,3)],[D(20
25,2,18),R(7512482,3)],[D(2025,2,19),R(7514305,3)],[D(2025,2,20),R(7492602,3)],[D(2025,2,21),R(7497618,3)],[D(2025,2,24),R(7540058,3)],[D(2
025,2,25),R(7478657,3)],[D(2025,2,26),R(7497187,3)],[D(2025,2,27),R(7400899,3)],[D(2025,2,28),R(7367585,3)],[D(2025,3,3),R(7444628,3)],[D(2
025,3,4),R(7496334,3)],[D(2025,3,5),R(7466535,3)],[D(2025,3,6),R(7479139,3)],[D(2025,3,7),R(7477204,3)],[D(2025,3,10),R(7433074,3)],[D(202
5,3,11),R(7497898,3)],[D(2025,3,12),R(7541758,3)],[D(2025,3,13),R(7649151,3)],[D(2025,3,14),R(7653370,3)],[D(2025,3,17),R(7685280,3)],[D(20
25,3,18),R(7745908,3)],[D(2025,3,19),R(7756515,3)],[D(2025,3,20),R(7745853,3)],[D(2025,3,21),R(7661014,3)],[D(2025,3,24),R(7589103,3)],[D(2
025,3,25),R(7619642,3)],[D(2025,3,26),R(7629515,3)],[D(2025,3,27),R(7718155,3)],[D(2025,3,28),R(7773123,3)],[D(2025,3,31),R(7867160,3)],[D
(2025,4,1),R(7853178,3)],[D(2025,4,2),R(7941385,3)],[D(2025,4,3),R(7826433,3)],[D(2025,4,4),R(7661132,3)],[D(2025,4,7),R(7566580,3)],[D(202
5,4,8),R(7602489,3)],[D(2025,4,9),R(7825457,3)],[D(2025,4,10),R(8068740,3)],[D(2025,4,11),R(8207242,3)],[D(2025,4,14),R(8140798,3)],[D(202
5,4,15),R(8167083,3)],[D(2025,4,16),R(8432597,3)],[D(2025,4,17),R(8352864,3)],[D(2025,4,18),R(8352864,3)],[D(2025,4,21),R(8615461,3)],[D(20
25,4,22),R(8348239,3)],[D(2025,4,23),R(8342062,3)],[D(2025,4,24),R(8448257,3)],[D(2025,4,25),R(8350805,3)],[D(2025,4,28),R(8370777,3)],[D(2
025,4,29),R(8330619,3)],[D(2025,4,30),R(8167491,3)],[D(2025,5,1),R(8083632,3)],[D(2025,5,2),R(8084630,3)],[D(2025,5,5),R(8266986,3)],[D(202
5,5,6),R(8486622,3)],[D(2025,5,7),R(8412541,3)],[D(2025,5,8),R(8390680,3)],[D(2025,5,9),R(8377691,3)],[D(2025,5,12),R(8099199,3)],[D(202
5,5,13),R(8147983,3)],[D(2025,5,14),R(8012772,3)],[D(2025,5,15),R(8175991,3)],[D(2025,5,16),R(8074356,3)],[D(2025,5,19),R(8119627,3)],[D(20
25,5,20),R(8309978,3)],[D(2025,5,21),R(8374651,3)],[D(2025,5,22),R(8359907,3)],[D(2025,5,23),R(8418063,3)],[D(2025,5,26),R(8399407,3)],[D(2
025,5,27),R(8313371,3)],[D(2025,5,28),R(8268390,3)],[D(2025,5,29),R(8347140,3)],[D(2025,5,30),R(8295795,3)],[D(2025,6,2),R(8529179,3)],[D(2
025,6,3),R(8481813,3)],[D(2025,6,4),R(8553307,3)],[D(2025,6,5),R(8509104,3)],[D(2025,6,6),R(8373064,3)],[D(2025,6,9),R(8404074,3)],[D(202
5,6,10),R(8386329,3)],[D(2025,6,11),R(8473836,3)],[D(2025,6,12),R(8542978,3)],[D(2025,6,13),R(8714514,3)],[D(2025,6,16),R(8596832,3)],[D(20
25,6,17),R(8657839,3)],[D(2025,6,18),R(8618259,3)],[D(2025,6,19),R(8605559,3)],[D(2025,6,20),R(8594988,3)],[D(2025,6,23),R(8542009,3)],[D(2
025,6,24),R(8418991,3)],[D(2025,6,25),R(8445181,3)],[D(2025,6,26),R(8389573,3)],[D(2025,6,27),R(8251201,3)],[D(2025,6,30),R(8353427,3)],[D
(2025,7,1),R(8433583,3)],[D(2025,7,2),R(8496357,3)],[D(2025,7,3),R(8369923,3)],[D(2025,7,4),R(8419905,3)],[D(2025,7,7),R(8461032,3)],[D(202
5,7,8),R(8339933,3)],[D(2025,7,9),R(8386782,3)],[D(2025,7,10),R(8402866,3)]]',
 'ChartDataName': 'Gold Price in Indian Rupees'}
```


In [19]:

```python
# Extract the dates and prices for 2024
chart_data_2025 = data_2025['ChartData']

pattern = r"D\((\d+),(\d+),(\d+)\),R\((\d+),(\d+)\)"

matches_2025 = re.findall(pattern, chart_data_2025)
```

```
parsed_data_2025 = []
for year, month, day, value, decimal in matches_2025:
    date = datetime(int(year), int(month), int(day))
    price = int(value) / (10 ** int(decimal)) # e.g: divide by 1000
    parsed_data_2025.append((date, round(price, 2)))

df_2025 = pd.DataFrame(parsed_data_2025, columns=["Date", "gold_rate"])
display(df_2025)
```

| | Date | gold_rate |
|---|---|---|
| **0** | 2025-01-01 | 6617.80 |
| **1** | 2025-01-02 | 6723.70 |
| **2** | 2025-01-03 | 6676.08 |
| **3** | 2025-01-06 | 6653.73 |
| **4** | 2025-01-07 | 6698.45 |
| **...** | ... | ... |
| **132** | 2025-07-04 | 8419.91 |
| **133** | 2025-07-07 | 8461.03 |
| **134** | 2025-07-08 | 8339.93 |
| **135** | 2025-07-09 | 8386.78 |
| **136** | 2025-07-10 | 8402.87 |

137 rows × 2 columns

In [20]:
```
# Append the 2024 & 2025
gold_data = pd.concat([df_2024, df_2025])
display(gold_data)
```

| | Date | gold_rate |
|---|---|---|
| **0** | 2024-01-01 | 5066.31 |
| **1** | 2024-01-02 | 5053.34 |
| **2** | 2024-01-03 | 5014.55 |
| **3** | 2024-01-04 | 5012.48 |
| **4** | 2024-01-05 | 5015.20 |
| **...** | ... | ... |
| **132** | 2025-07-04 | 8419.91 |
| **133** | 2025-07-07 | 8461.03 |
| **134** | 2025-07-08 | 8339.93 |
| **135** | 2025-07-09 | 8386.78 |
| **136** | 2025-07-10 | 8402.87 |

399 rows × 2 columns

In [21]:
```
# Merge the 2 dataframes
gold_dataset = pd.merge(usd_inr, gold_data, on='Date', how='inner')
display(gold_dataset)
```

| | Date | USD_INR | gold_rate |
|---|---|---|---|
| **0** | 2024-01-01 | 83.248596 | 5066.31 |
| **1** | 2024-01-02 | 83.202599 | 5053.34 |
| **2** | 2024-01-03 | 83.257004 | 5014.55 |
| **3** | 2024-01-04 | 83.318100 | 5012.48 |
| **4** | 2024-01-05 | 83.240601 | 5015.20 |
| **...** | ... | ... | ... |
| **387** | 2025-06-30 | 85.455299 | 8353.43 |
| **388** | 2025-07-01 | 85.713997 | 8433.58 |
| **389** | 2025-07-02 | 85.642799 | 8496.36 |
| **390** | 2025-07-03 | 85.694099 | 8369.92 |
| **391** | 2025-07-04 | 85.319099 | 8419.91 |

392 rows × 3 columns

# Exploratory Data Analysis

-- Handle Missing values - None

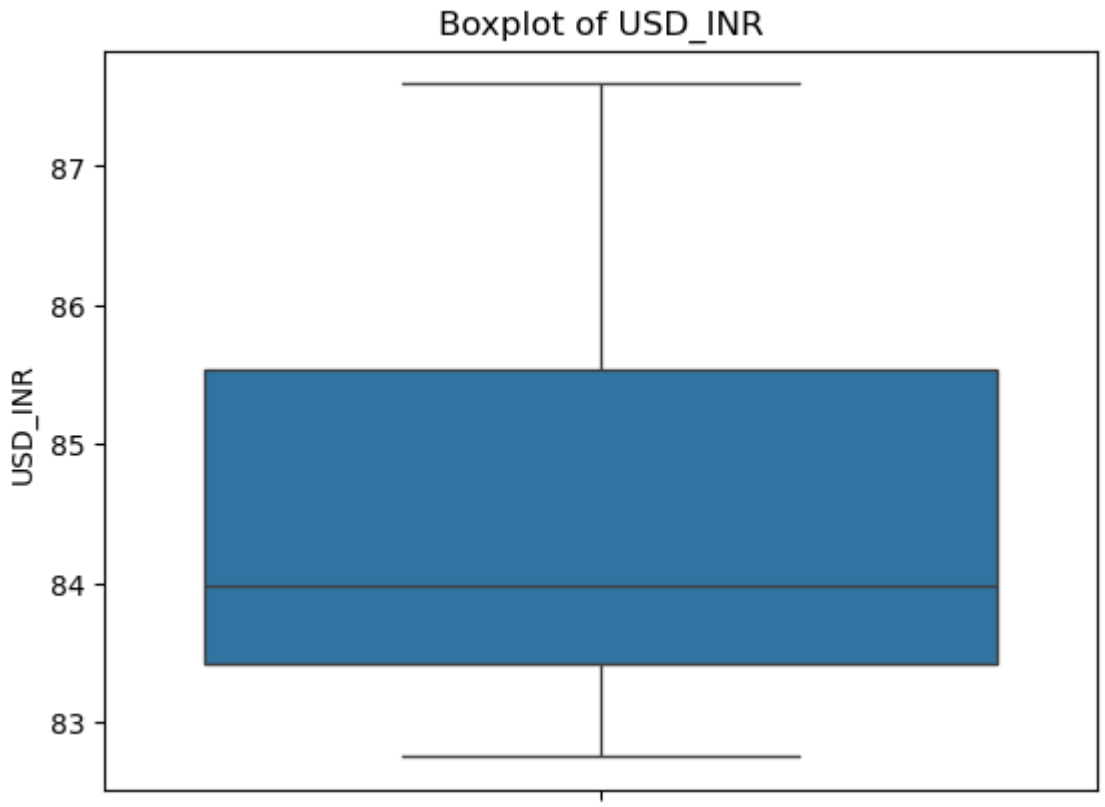
-- Handle imbalanced dataset - None

-- Hande outliers - Noted

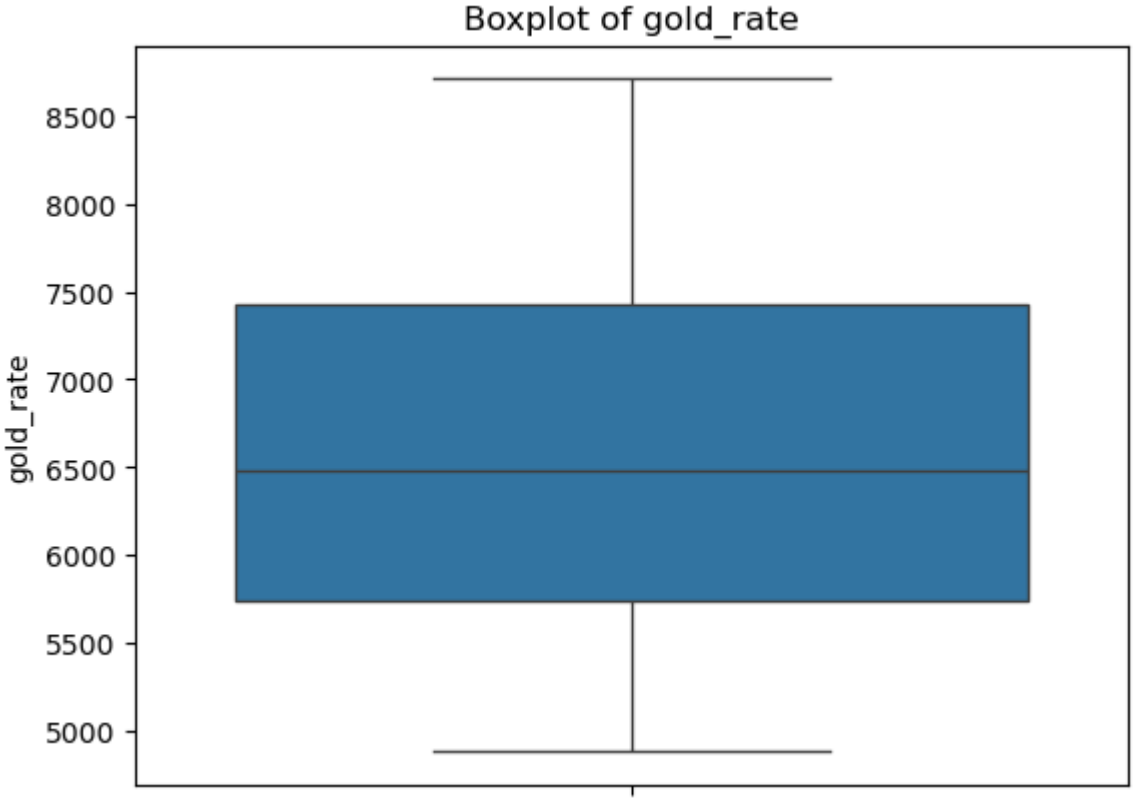
-- Encode catagorical features - None

-- Normalization or Standardisation - Standardisation

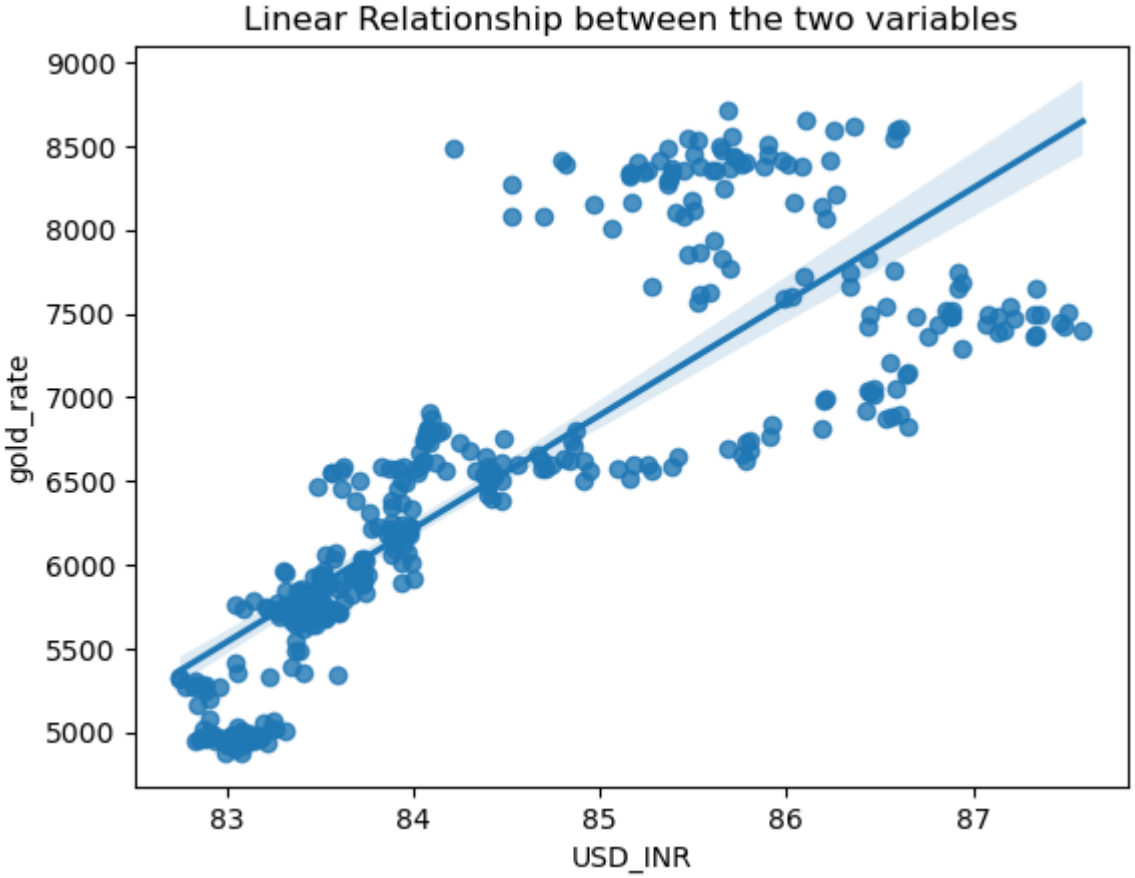
In [25]:
```
# Visualize the variable in boxplot
sns.boxplot(data=gold_dataset, y= 'USD_INR')
plt.title("Boxplot of USD_INR")
plt.show()
```




In [26]:
```
# Visualize the variable in boxplot
sns.boxplot(data=gold_dataset, y= 'gold_rate')
plt.title("Boxplot of gold_rate")
plt.show()
```




In [27]:
```
# visualize the relationship between the two variables
sns.regplot(data= gold_dataset, x = 'USD_INR', y= 'gold_rate')
plt.title("Linear Relationship between the two variables")
plt.show()
```




**Insight:** As the USD/INR increases, the gold rate also tends to rise, showing a positive relationship between the two variables.

## Model training

So basically, I am going to teaching the model: "If I know the USD to INR rate, can I predict the gold rate in INR?"

In [28]:
```
# Assigning the columns to X,y variables
X = gold_dataset[['USD_INR']]
y = gold_dataset[['gold_rate']]
```

In [29]:
```
type(X)
```

Out[29]:
```
pandas.core.frame.DataFrame
```

In [30]:
```
X.head()
```

Out[30]:

| | USD_INR |
|---|---|
| **0** | 83.248596 |
| **1** | 83.202599 |
| **2** | 83.257004 |
| **3** | 83.318100 |
| **4** | 83.240601 |

In [32]:
```
# Splits data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.1, random_state= 42)
```

X_train 90% of the input data for training the model

X_test 10% of the input data to test the model

y_train 90% of the output data (gold rates) for training

y_test 10% of the output data to check predictions

test_size=0.1 =10% of the full dataset will be used for testing

random_state=42 Keeps the split the same every time you run it

In [33]:
```
len(X_train)
```

Out[33]: 352

In [35]:
```
# Transform the values using standardisation
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

fit_transform(X_train) → calculates mean & std from training data, then applies scaling

transform(X_test) → uses the same mean & std from training to scale test data

In [36]:
```
X_train_scaled[:5]
```

Out[36]:
```
array([[-0.96891716],
       [-0.57899468],
       [ 0.94562025],
       [-0.6889582 ],
       [-0.42133368]])
```

## Using linear regression

In [38]:
```
# Build the model to learn the releationship between the two variables
regressor = LinearRegression()

regressor.fit(X_train_scaled, y_train)
```

Out[38]:
```
▾ LinearRegression

LinearRegression()
```

In [ ]:
```
# Print the model's parameters
regressor.get_params()
```

Out[ ]:
```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'positive': False}
```

In [ ]:
```
# Print the slope(m) value
regressor.coef_
```

Out[ ]:
```
array([[874.92586566]])
```

In [41]:
```
# Print the intercept(b)
regressor.intercept_
```

Out[41]:
```
array([6540.76110795])
```

In [42]:
```
# y = mx + b
m = regressor.coef_[0][0]
b = regressor.intercept_[0]
```

In [43]:
```
m, b
```

Out[43]:
```
(np.float64(874.9258656619443), np.float64(6540.761107954547))
```

In [44]:
```
# Predic the gold_rate from the training
X_train_predict = regressor.predict(X_train_scaled)
```

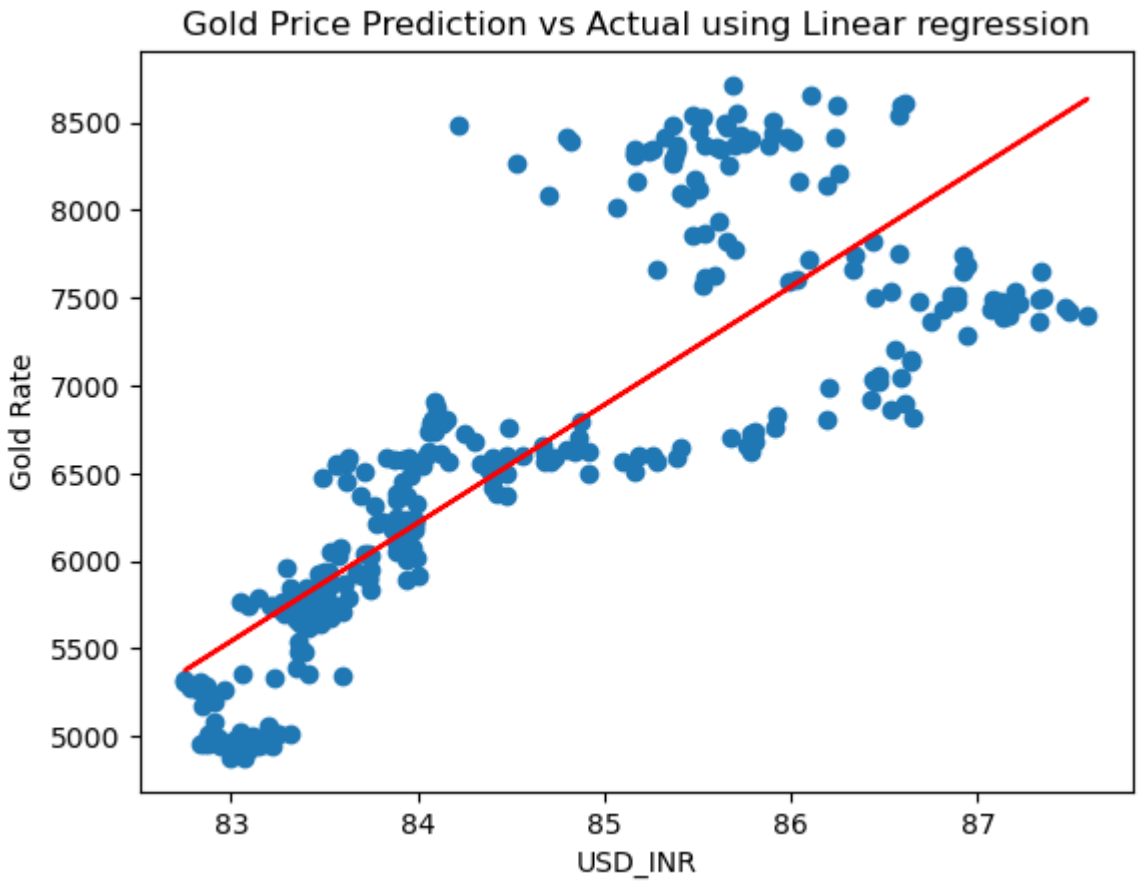
In [45]:
```
X_train_predict[:5]
```

Out[45]:
```
array([[5693.0304213 ],
       [6034.18368246],
       [7368.10872511],
       [5937.97376136],
       [6172.12537696]])
```

## Visualize actual vs prediction

In [71]:
```
plt.scatter(X_train, y_train)   # Scatter plot (dots) of actual data
plt.plot(X_train, X_train_predict, color = "Red")   # Line plot of model predictions
plt.xlabel("USD_INR")
plt.ylabel("Gold Rate")
plt.title("Gold Price Prediction vs Actual using Linear regression")

plt.show()
```



Dots = Real values

Line = Model's predicted values

In [47]:
```
# Predict the test data
X_test_predict = regressor.predict(X_test_scaled)
```

In [48]:
```
X_test_predict
```

Out[48]:
```
array([[5943.5033628 ],
       [7948.06361089],
       [6787.27424692],
       [5466.90315754],
       [7197.19517463],
       [6177.04800819],
       [5506.88088994],
       [6474.30909352],
       [6032.49651105],
       [5899.00421676],
       [5593.1169532 ],
       [6242.51334533],
       [8463.90083914],
       [6141.72028481],
       [7706.76209205],
       [7024.59959655],
       [5947.41266241],
       [7619.8573328 ],
       [7857.45016083],
       [7705.54815164],
       [5625.88305656],
       [7371.61166332],
       [6574.15569203],
       [5709.41347298],
       [5892.60016673],
       [5798.54550424],
       [5991.84493885],
       [5570.86480827],
       [5754.18524122],
       [5882.95551307],
       [6037.55288147],
       [6853.14594547],
       [8584.58503336],
       [6867.10111633],
       [5798.00540364],
       [7808.97484558],
       [5874.19045183],
       [5570.39157727],
       [5820.05179595],
       [5541.33416472]])
```

In [ ]:
```
# Calculate evaluation metrics for X_train_Lr model
mae = mean_absolute_error(y_train, X_train_predict)
mse = mean_squared_error(y_train, X_train_predict)
r2 = r2_score(y_train, X_train_predict)
```

```
print("Simple Linear Regression: ")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R² Score: {r2:.4f}")
```

```
Simple Linear Regression:
Mean Absolute Error (MAE): 463.97
Mean Squared Error (MSE): 373678.68
R² Score: 0.6720
```

In [ ]:

```
# Calculate evaluation metrics for X_test_Lr model
mae = mean_absolute_error(y_test, X_test_predict)
mse = mean_squared_error(y_test, X_test_predict)
r2 = r2_score(y_test, X_test_predict)

print("Simple Linear Regression: ")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R² Score: {r2:.4f}")
```

```
Simple Linear Regression:
Mean Absolute Error (MAE): 484.32
Mean Squared Error (MSE): 419475.23
R² Score: 0.6769
```

| Metric | Ideal Value | Meaning |
|---|---|---|
| MAE | Close to 0 | Smaller = Better (fewer ₹ errors) |
| MSE | Close to 0 | Lower = Better (no large errors) |
| $R^2$ | Close to 1 | Higher = Better (model fits well) |

## Using Random Forest Regressor

In [52]:

```
# Build the model to learn the releationship between the two variables
rfr_model = RandomForestRegressor()
rfr_model.fit(X_train_scaled, y_train)
```

```
c:\Users\sanka\anaconda3\envs\learning\Lib\site-packages\sklearn\base.py:1389: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

Out[52]:

```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [53]:

```
X_train_predict_rfr = rfr_model.predict(X_train_scaled)
```

In [54]:

```
X_train_predict_rfr[0]
```

Out[54]:

```
np.float64(5617.636800000008)
```

In [79]:

```
X_test_predict_rfr = rfr_model.predict(X_test_scaled)
```

In [80]:

```
X_test_predict_rfr[0]
```

Out[80]:

```
np.float64(5718.750000000001)
```

In [ ]:

```
# Calculate evaluation metrics for X_train_rfr model
mae = mean_absolute_error(y_train, X_train_predict_rfr)
mse = mean_squared_error(y_train, X_train_predict_rfr)
r2 = r2_score(y_train, X_train_predict_rfr)

print("RandomForestRegressor:")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R² Score: {r2:.4f}")
```

```
RandomForestRegressor:
Mean Absolute Error (MAE): 121.61
Mean Squared Error (MSE): 37145.53
R² Score: 0.9674
```

In [ ]:
```
# Calculate evaluation metrics for X_test_rfr_model
mae = mean_absolute_error(y_test, X_test_predict_rfr)
mse = mean_squared_error(y_test, X_test_predict_rfr)
r2 = r2_score(y_test, X_test_predict_rfr)

print("RandomForestRegressor:")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R² Score: {r2:.4f}")
```

```
RandomForestRegressor:
Mean Absolute Error (MAE): 262.26
Mean Squared Error (MSE): 186641.77
R² Score: 0.8563
```

**No severe overfitting**

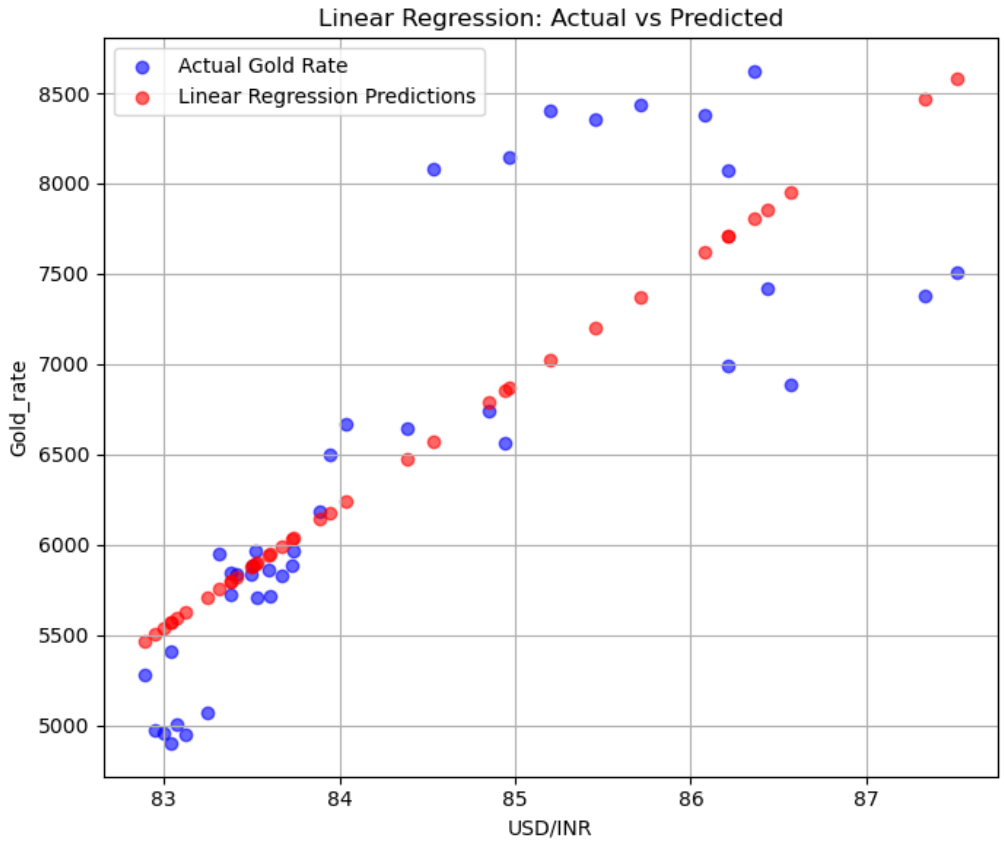
# Comparing the models using plots

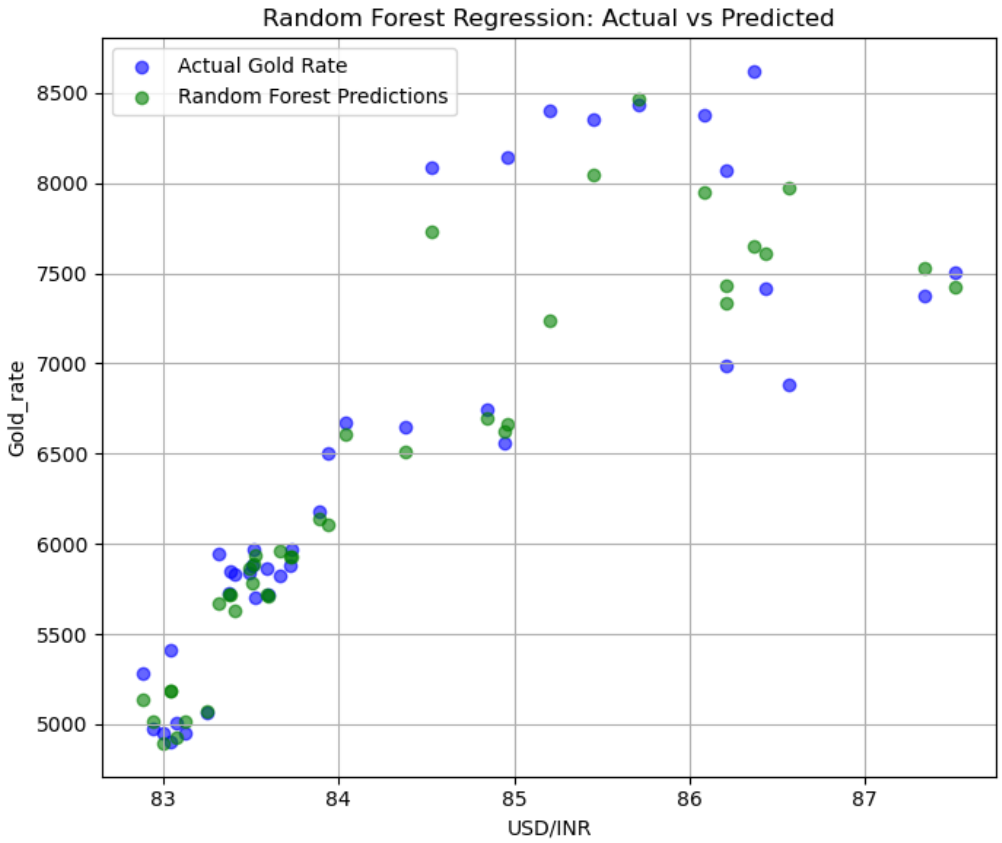
In [83]:
```
# Create a figure with two subplots
plt.figure(figsize=(14, 6))

# Plot 1: Linear Regression
plt.subplot(1, 2, 1)
plt.scatter(X_test, y_test, color='blue', label='Actual Gold Rate', alpha=0.6)
plt.scatter(X_test, X_test_predict, color='red', label='Linear Regression Predictions', alpha=0.6)
plt.xlabel('USD/INR')
plt.ylabel('Gold_rate')
plt.title('Linear Regression: Actual vs Predicted')
plt.legend()
plt.grid(True)

# Plot 2: Random Forest Regression
plt.subplot(1, 2, 2)
plt.scatter(X_test, y_test, color='blue', label='Actual Gold Rate', alpha=0.6)
plt.scatter(X_test, X_test_predict_rfr, color='green', label='Random Forest Predictions', alpha=0.6)
plt.xlabel('USD/INR')
plt.ylabel('Gold_rate')
plt.title('Random Forest Regression: Actual vs Predicted')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```




Random Forest's predictions closely follow actual gold rates with higher accuracy ($R^2$: 0.85 vs 0.67) → Deploy Random Forest.

Why?

-- Lower errors (MAE: 262 vs 484) and better fit for non-linear trends.

-- Visual plots confirm RF predictions align tighter with actual data points.

# MLOPS

-- Save the model

-- Build interface

-- Connect model & Interface

In [86]:
```
# Save the dataframes
```

```
gold_dataset.to_csv("gold_dataset_for_price_predictions.csv", index=False)
```

In [90]:
```
#Save the model using pickle for later use
pickle.dump(regressor, open('regressor.pkl', 'wb'))
pickle.dump(rfr_model, open('rfr_model.pkl', 'wb'))
pickle.dump(scaler, open('scaler.pkl', 'wb'))
```

In [91]:
```
# Reload the model
regressor_reloaded = pickle.load(open('regressor.pkl', 'rb'))
rfr_reloaded = pickle.load(open('rfr_model.pkl', 'rb'))
scaler_reloded = pickle.load(open('scaler.pkl', 'rb'))
```

## Creating the demo interface

In [95]:
```
# Prediction function
def predict_gold_price(usd_inr):
    scaled_input = scaler_reloded.transform(np.array(usd_inr).reshape(1, -1))
    prediction = rfr_model.predict(scaled_input)

    return prediction[0].round(2)

# Create input and output interface
import gradio as gr

interface = gr.Interface(fn=predict_gold_price, inputs="number", outputs="number", title="Gold Price Predictor")

interface.launch()
```

```
* Running on local URL:  http://127.0.0.1:7861
* To create a public link, set `share=True` in `launch()`.
```

# Unable to connect

Firefox can’t establish a connection to the server at 127.0.0.1:7861.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer’s network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the web.

Try Again

Out[95]:

```
c:\Users\sanka\anaconda3\envs\learning\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature names,
but StandardScaler was fitted with feature names
  warnings.warn(
```

In [ ]: