Gold Price INR Prediction 🥇

Author

DINESH S
my linkedin profile | github link

Overview

This notebook focuses on predicting **gold prices in INR** using machine learning and time-series models. The workflow includes **data collection, preprocessing, feature engineering, model training, evaluation, visualization, and deployment** using Gradio. Below is an in-depth breakdown of its key components.

In [ ]:
```
!pip install gradio
```

```
Collecting gradio
  Downloading gradio-5.15.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<24.0,>=22.0 (from gradio)
  Downloading aiofiles-23.2.1-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/
dist-packages (from gradio) (3.7.1)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.8-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.7.0 (from gradio)
  Downloading gradio_client-1.7.0-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/di
st-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/pyt
hon3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-
packages (from gradio) (3.1.5)
Collecting markupsafe~=2.0 (from gradio)
  Downloading MarkupSafe-2.1.5-cp311-cp311-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl.metadata (3.0 kB)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/
dist-packages (from gradio) (1.26.4)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist
-packages (from gradio) (3.10.15)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-p
ackages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.1
1/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.1
1/dist-packages (from gradio) (11.1.0)
Requirement already satisfied: pydantic>=2.0 in /usr/local/lib/python3.11/di
st-packages (from gradio) (2.10.6)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.1
1/dist-packages (from gradio) (6.0.2)
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.9.5-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.45.3-py3-none-any.whl.metadata (6.3 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.1
1/dist-packages (from gradio) (0.15.1)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/pyth
on3.11/dist-packages (from gradio) (4.12.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.0-py3-none-any.whl.metadata (6.5 kB)
```

```
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.7.0->gradio) (2024.10.0)
Requirement already satisfied: websockets<15.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.7.0->gradio) (14.2)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.1.31)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.7)
Requirement already satisfied: h11<0.15,>=0.13 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.14.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.17.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.1)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2.0->gradio) (0.7.0)
Requirement already satisfied: pydantic-core==2.27.2 in /usr/local/lib/python3.11/dist-packages (from pydantic>=2.0->gradio) (2.27.2)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0->gradio) (1.17.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (2.18.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gradio) (3.4.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gradio) (2.3.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0,>=0.12->gradio) (0.1.2)
Downloading gradio-5.15.0-py3-none-any.whl (57.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.8/57.8 MB 14.5 MB/s eta 0:00:00
Downloading gradio_client-1.7.0-py3-none-any.whl (321 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 321.9/321.9 kB 23.8 MB/s eta 0:0
```

```
0:00
Downloading aiofiles-23.2.1-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.8-py3-none-any.whl (94 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.8/94.8 kB 7.5 MB/s eta 0:00:0
0
Downloading MarkupSafe-2.1.5-cp311-cp311-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (28 kB)
Downloading python_multipart-0.0.20-py3-none-any.whl (24 kB)
Downloading ruff-0.9.5-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl (12.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 88.6 MB/s eta 0:00:
00
Downloading safehttpx-0.1.6-py3-none-any.whl (8.7 kB)
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading starlette-0.45.3-py3-none-any.whl (71 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.5/71.5 kB 5.8 MB/s eta 0:00:0
0
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading uvicorn-0.34.0-py3-none-any.whl (62 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.3/62.3 kB 5.1 MB/s eta 0:00:0
0
Downloading ffmpy-0.5.0-py3-none-any.whl (6.0 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ru
ff, python-multipart, markupsafe, ffmpy, aiofiles, starlette, safehttpx, gra
dio-client, fastapi, gradio
  Attempting uninstall: markupsafe
    Found existing installation: MarkupSafe 3.0.2
    Uninstalling MarkupSafe-3.0.2:
      Successfully uninstalled MarkupSafe-3.0.2
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
torch 2.5.1+cu124 requires nvidia-cublas-cu12==12.4.5.8; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cublas-cu12 1
2.5.3.2 which is incompatible.
torch 2.5.1+cu124 requires nvidia-cuda-cupti-cu12==12.4.127; platform_system
== "Linux" and platform_machine == "x86_64", but you have nvidia-cuda-cupti-
cu12 12.5.82 which is incompatible.
torch 2.5.1+cu124 requires nvidia-cuda-nvrtc-cu12==12.4.127; platform_system
== "Linux" and platform_machine == "x86_64", but you have nvidia-cuda-nvrtc-
cu12 12.5.82 which is incompatible.
torch 2.5.1+cu124 requires nvidia-cuda-runtime-cu12==12.4.127; platform_syst
em == "Linux" and platform_machine == "x86_64", but you have nvidia-cuda-run
time-cu12 12.5.82 which is incompatible.
torch 2.5.1+cu124 requires nvidia-cudnn-cu12==9.1.0.70; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cudnn-cu12 9.
3.0.75 which is incompatible.
torch 2.5.1+cu124 requires nvidia-cufft-cu12==11.2.1.3; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cufft-cu12 11.
2.3.61 which is incompatible.
torch 2.5.1+cu124 requires nvidia-curand-cu12==10.3.5.147; platform_system =
= "Linux" and platform_machine == "x86_64", but you have nvidia-curand-cu12
10.3.6.82 which is incompatible.
torch 2.5.1+cu124 requires nvidia-cusolver-cu12==11.6.1.9; platform_system =
= "Linux" and platform_machine == "x86_64", but you have nvidia-cusolver-cu1
```

```
2 11.6.3.83 which is incompatible.
torch 2.5.1+cu124 requires nvidia-cusparse-cu12==12.3.1.170; platform_system
== "Linux" and platform_machine == "x86_64", but you have nvidia-cusparse-cu
12 12.5.1.3 which is incompatible.
torch 2.5.1+cu124 requires nvidia-nvjitlink-cu12==12.4.127; platform_system
== "Linux" and platform_machine == "x86_64", but you have nvidia-nvjitlink-c
u12 12.5.82 which is incompatible.
Successfully installed aiofiles-23.2.1 fastapi-0.115.8 ffmpy-0.5.0 gradio-5.
15.0 gradio-client-1.7.0 markupsafe-2.1.5 pydub-0.25.1 python-multipart-0.0.
20 ruff-0.9.5 safehttpx-0.1.6 semantic-version-2.10.0 starlette-0.45.3 tomlk
it-0.13.2 uvicorn-0.34.0
```

In [209…

```
# Initializing all the Needed Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import gradio as gr
```

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
from sklearn.linear_model import Ridge
```

In [ ]:

```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
```

1.Data Collection & Preprocessing

Data Source

- The dataset consists of **53 weekly data points from 2024**, including **Date, USD/INR exchange rate, and Gold Rate in INR**.
- Retrieved from Exchange Rates.

In [ ]:

```
gold_usdinr = pd.read_csv('/content/Gold vs USDINR.csv')
```

In [ ]:

```
gold_usdinr.head(5)
```

Out[ ]:

	Date	USD_INR	Goldrate
0	2024-01-01	83.240601	₹5,066.31
1	2024-01-08	83.076103	₹4,966.31
2	2024-01-15	83.160599	₹5,015.33
3	2024-01-22	83.146103	₹4,950.84
4	2024-01-29	82.927597	₹4,976.77

In [ ]:

```
gold_usdinr.describe()
```

Out[ ]:

	USD_INR
count	53.000000
mean	83.717398
std	0.637302
min	82.752296
25%	83.301804
50%	83.544998
75%	83.988998
max	85.786598

In [ ]:

```
gold_usdinr["Goldrate"] = gold_usdinr["Goldrate"].replace("₹",'',regex=True)
```

In [ ]:

```
gold_usdinr.head(5)
```

Out[ ]:

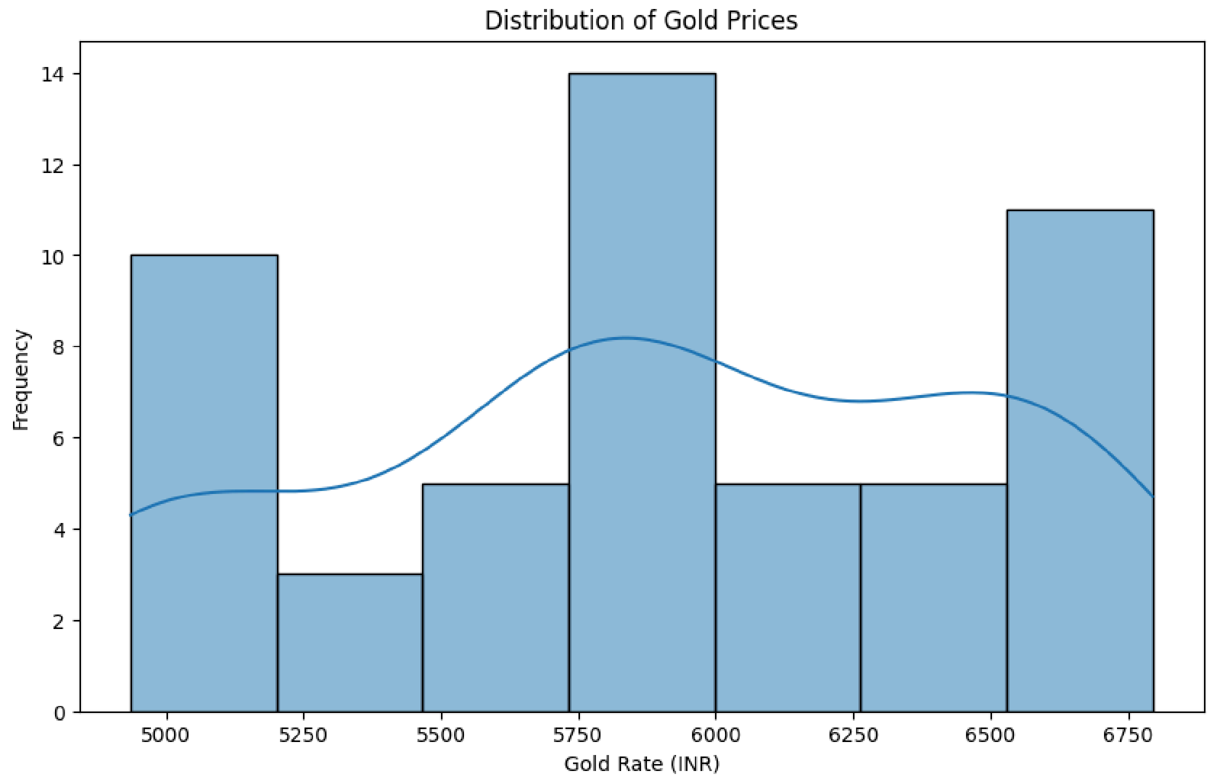
	Date	USD_INR	Goldrate
0	2024-01-01	83.240601	5066.31
1	2024-01-08	83.076103	4966.31
2	2024-01-15	83.160599	5015.33
3	2024-01-22	83.146103	4950.84
4	2024-01-29	82.927597	4976.77

2.Exploratory Data Analysis (EDA)

In [ ]:

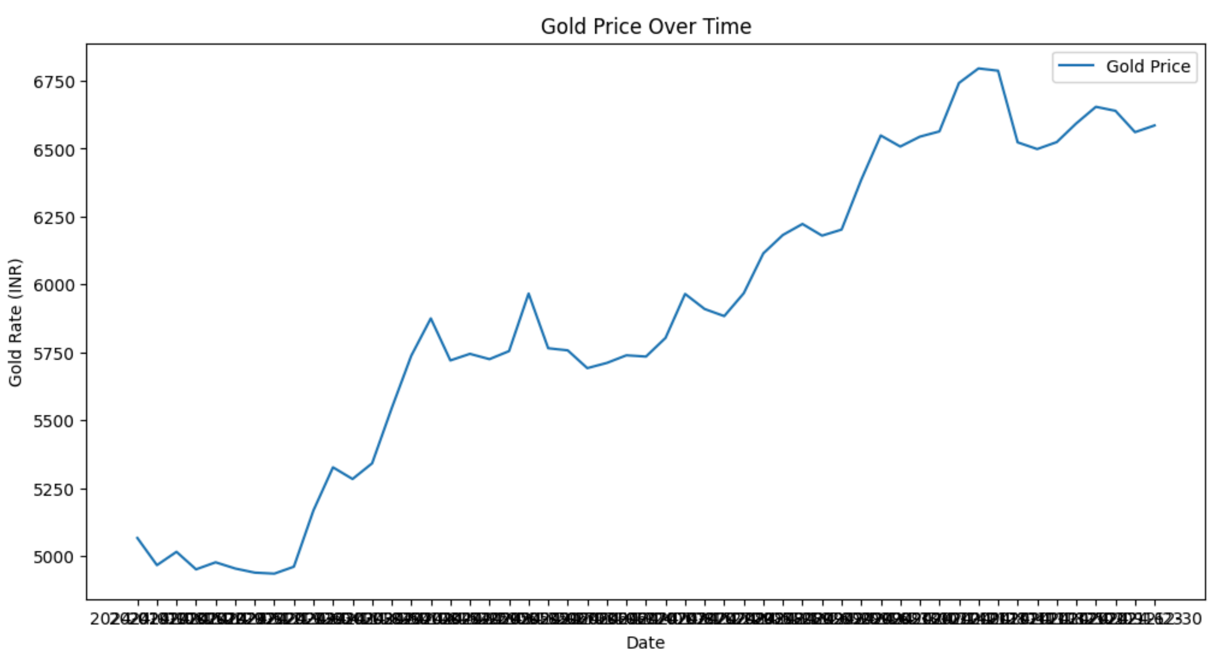
```
# Visualize the distribution of 'Goldrate'
plt.figure(figsize=(10, 6))
sns.histplot(gold_usdinr['Goldrate'], kde=True)
plt.title('Distribution of Gold Prices')
plt.xlabel('Gold Rate (INR)')
```

```python
plt.ylabel('Frequency')
plt.show()
```



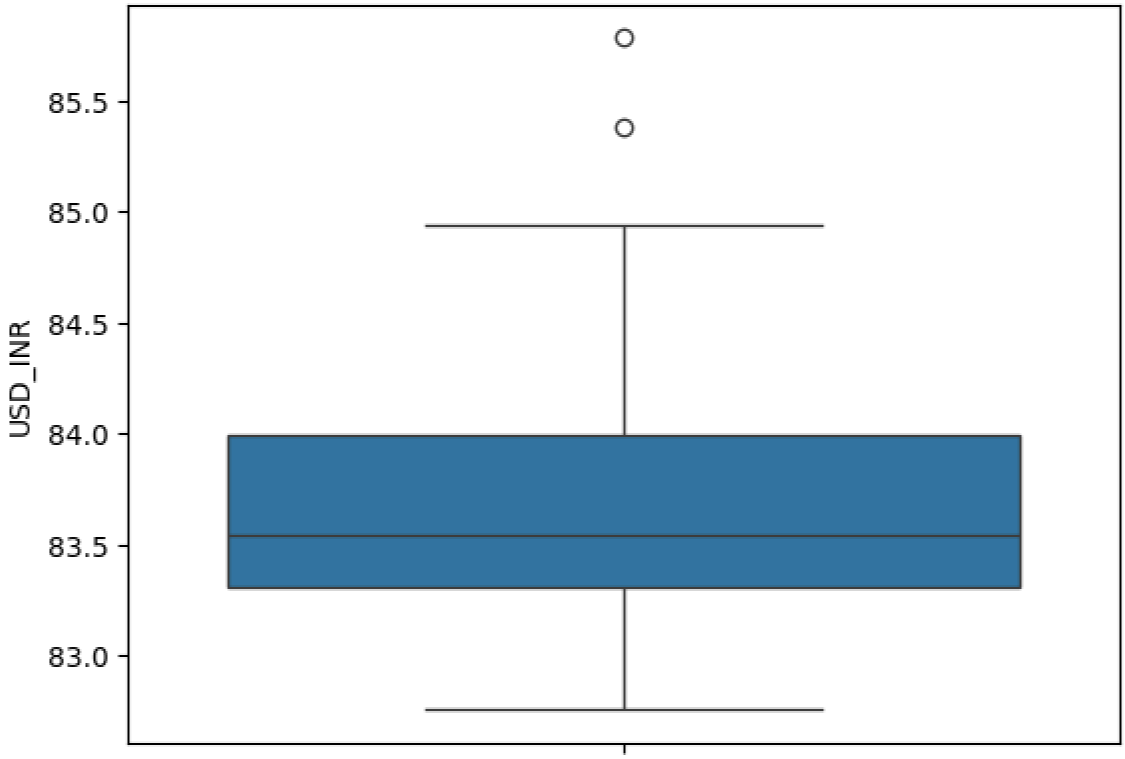
In [ ]:

```python
# Examine the relationship between gold prices and USDINR
plt.figure(figsize=(12,6))
sns.lineplot(data=gold_usdinr, x='Date', y='Goldrate', label='Gold Price')
plt.title('Gold Price Over Time')
plt.xlabel('Date')
plt.ylabel('Gold Rate (INR)')
plt.show()
```



In [ ]:
```
sns.boxplot(data=gold_usdinr['USD_INR'])
```

Out[ ]: <Axes: ylabel='USD_INR'>

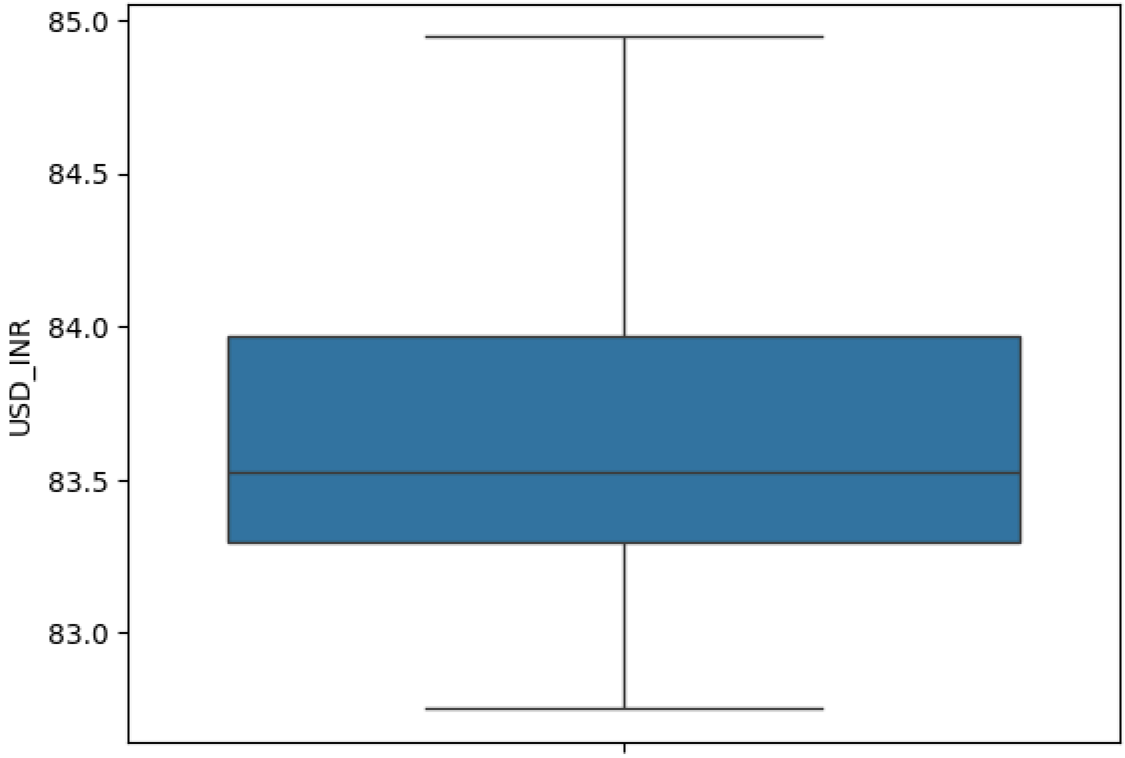


In [ ]:
```
#remove outliers

# Calculate IQR for USD_INR
Q1 = gold_usdinr['USD_INR'].quantile(0.25)
Q3 = gold_usdinr['USD_INR'].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers based on the bounds
gold_usdinr_cleaned = gold_usdinr[
    (gold_usdinr['USD_INR'] >= lower_bound) & (gold_usdinr['USD_INR'] <= upp
]

# Print the number of outliers removed
print(f"Number of outliers removed: {len(gold_usdinr) - len(gold_usdinr_clea
```

```
Number of outliers removed: 2
```

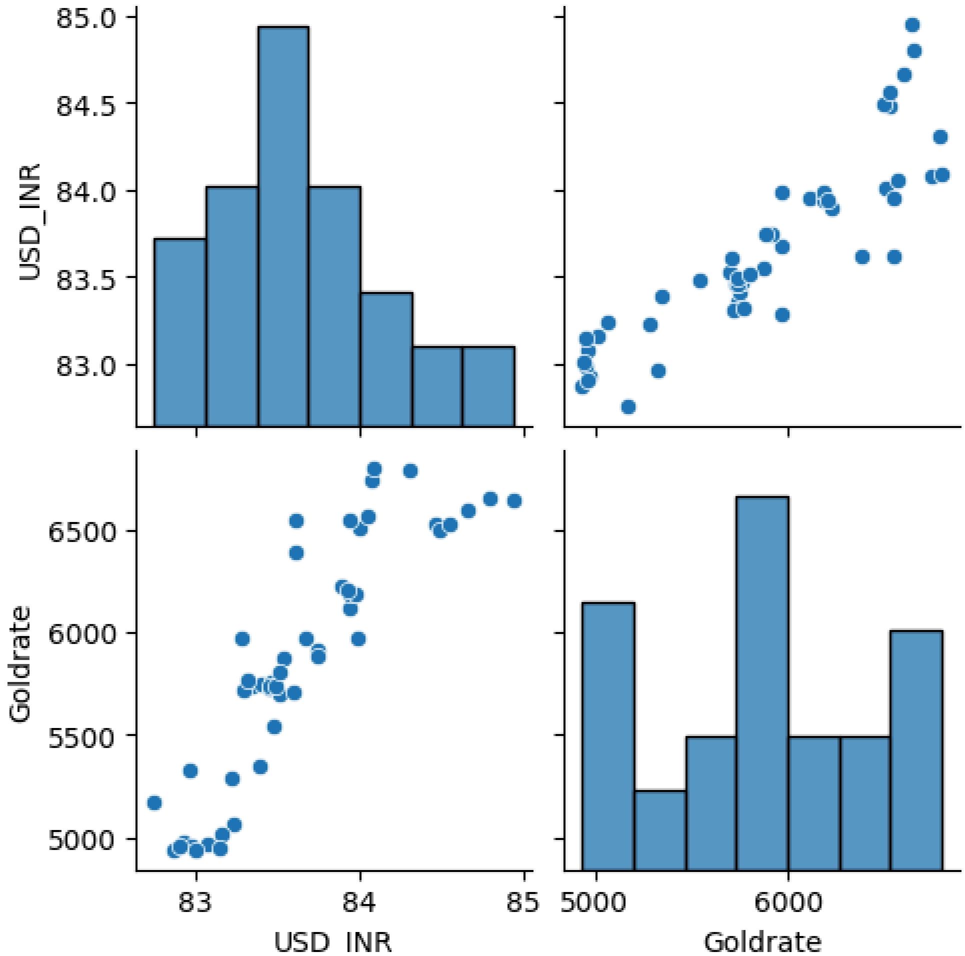
In [ ]:
```
sns.boxplot(data=gold_usdinr_cleaned['USD_INR'])
```

Out[ ]: <Axes: ylabel='USD_INR'>



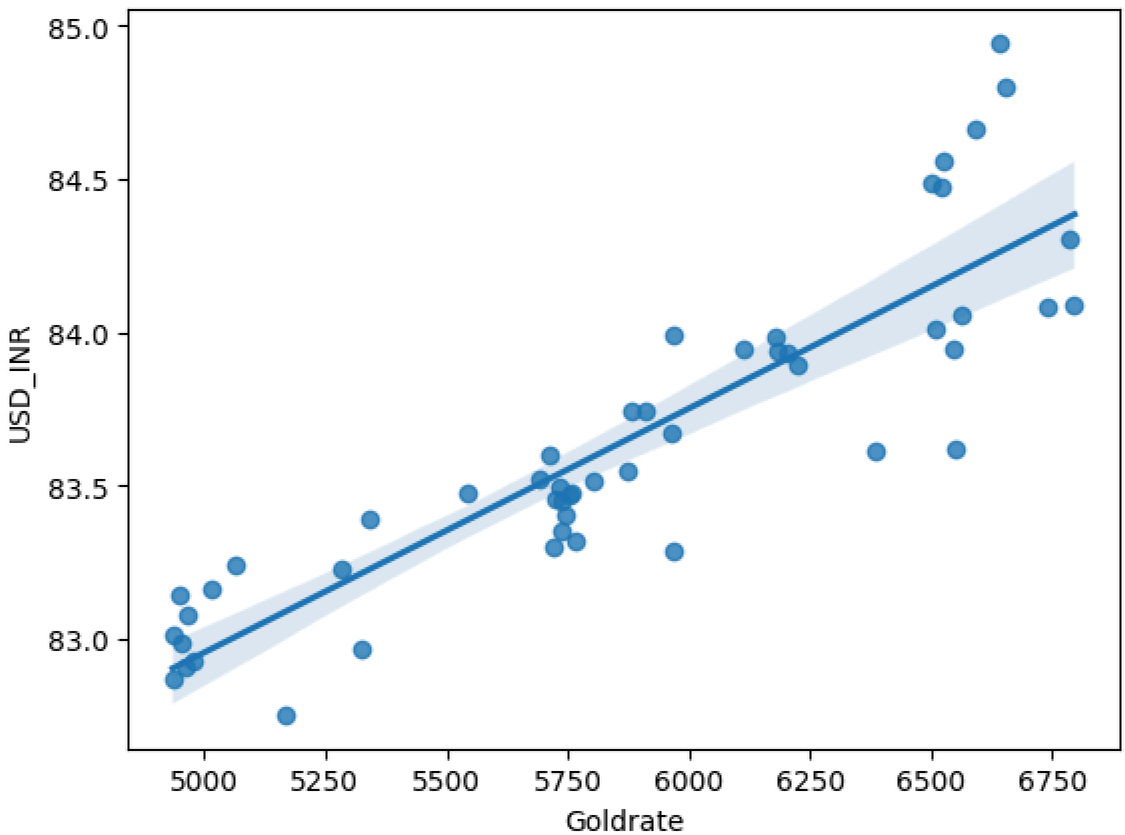
In [ ]:
```
sns.pairplot(gold_usdinr_cleaned)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x7ed4cd8d2f50>




```
In [ ]: sns.regplot(x='Goldrate',y='USD_INR',data=gold_usdinr_cleaned)
```

```
Out[ ]: <Axes: xlabel='Goldrate', ylabel='USD_INR'>
```



3.Feature Engineering

Techniques Used

- Extracted temporal features (e.g., day, month, year, day of the week) to capture seasonal trends.
- Standardized features before feeding them into machine learning models.
- Implemented **feature selection** to retain the most important attributes for prediction.

In [ ]:
```
X = gold_usdinr_cleaned[['USD_INR']]
y = gold_usdinr_cleaned[['Goldrate']]
```

In [ ]:
```
print(X.shape)
print(y.shape)
```

```
(51, 1)
(51, 1)
```

In [ ]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_st
```

In [ ]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(40, 1)
(11, 1)
(40, 1)
(11, 1)
```

In [ ]:
```
#standardization
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```
X_train_scaled
```

Out[ ]:
```
array([[-1.28057948e+00],
       [ 2.15516837e+00],
       [-1.10040558e+00],
       [ 1.71016575e+00],
       [-1.24934489e+00],
       [ 5.80053355e-01],
       [ 5.90584978e-01],
       [-2.63986902e-01],
       [ 1.90814640e+00],
       [-1.27948818e-01],
       [-1.56774633e+00],
       [-5.69667239e-01],
       [-2.21666410e-01],
       [ 1.25499179e+00],
       [-2.98865962e-01],
       [-7.06259609e-01],
       [-6.80831281e-01],
       [ 1.55905082e+00],
       [-1.76075567e-01],
       [ 5.07953312e-01],
       [-1.14454139e+00],
       [ 2.34228128e-01],
       [-1.34762630e-03],
       [-9.79610650e-01],
       [-5.34982180e-01],
       [-8.26139974e-01],
       [ 7.15550994e-01],
       [ 6.68685278e-01],
       [-1.69534864e-01],
       [ 7.96201058e-01],
       [-1.18032120e+00],
       [-2.58540948e-01],
       [-2.83428831e-01],
       [ 2.41489758e+00],
       [-5.97645330e-01],
       [-1.34978333e+00],
       [ 8.47057712e-01],
       [-4.83404938e-01],
       [ 1.06906367e-01],
       [ 3.01745541e-03]])
```

In [ ]:
```
X_test_scaled
```

Out[ ]:
```
array([[ 0.86467046],
       [ 0.60039603],
       [ 1.58537988],
       [-0.40402977],
       [-0.02514078],
       [ 0.67849631],
       [-0.37986245],
       [ 0.60458095],
       [-0.85246901],
       [ 0.23331354],
       [-0.25000479]])
```

4. Model Training & Evaluation

Models Implemented

1. **Linear Regression** – Basic baseline model.
2. **Random Forest Regressor** – Applied hyperparameter tuning with `RandomizedSearchCV`.
3. **ARIMA** – Time-series forecasting model.

1.Linear Regression Model

In [ ]:
```
lin_reg = LinearRegression()
```

In [ ]:
```
lin_reg.fit(X_train_scaled,y_train)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

In [ ]:
```
lin_reg.get_params()
```

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

In [ ]:
```
y_pred = lin_reg.predict(X_test_scaled)
print(y_pred)
print(y_test)
```

```
[[6303.91716248]
 [6165.06820038]
 [6682.57563976]
 [5637.34602291]
 [5836.4131137 ]
 [6206.10184462]
 [5650.04345372]
 [6167.26694494]
 [5401.73744228]
 [5972.20420554]
 [5718.27026614]]
    Goldrate
43   6795.35
40   6544.00
46   6498.43
12   5341.21
24   5710.88
31   5967.66
17   5744.34
32   6114.39
3    4950.84
30   5883.33
13   5543.85
```

In [ ]:

```
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,y_pred)
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared: {r2}")
```

```
Mean Squared Error: 75693.8314734317
Root Mean Squared Error: 275.1251196699998
R-squared: 0.7244658615036519
```

In [ ]:

```
#parameter tuning using randomizedsearchcv

# Define the parameter grid for RandomizedSearchCV
param_grid = {
    'alpha': np.logspace(-4, 4, 50),  # Explore a wide range of alpha values
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'
    'fit_intercept': [True, False]  # Include or exclude the intercept
}

# Create a Ridge regression model
ridge = Ridge()

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=ridge,
    param_distributions=param_grid,
    n_iter=10,  # Number of random parameter combinations to try
    cv=5,  # Number of cross-validation folds
    scoring='neg_mean_squared_error',  # Use negative MSE as the scoring met
    n_jobs=-1,  # Use all available CPU cores
    random_state=42,  # Set a random state for reproducibility
```

```
    verbose=2  # Set verbosity for output
)
```

In [ ]:
```
# Fit the RandomizedSearchCV object to your training data
random_search.fit(X_train_scaled, y_train)

# Print the best hyperparameters found
print("Best hyperparameters:", random_search.best_params_)

# Get the best model
best_ridge = random_search.best_estimator_

# Evaluate the best model on the test set
y_pred_best = best_ridge.predict(X_test_scaled)
```

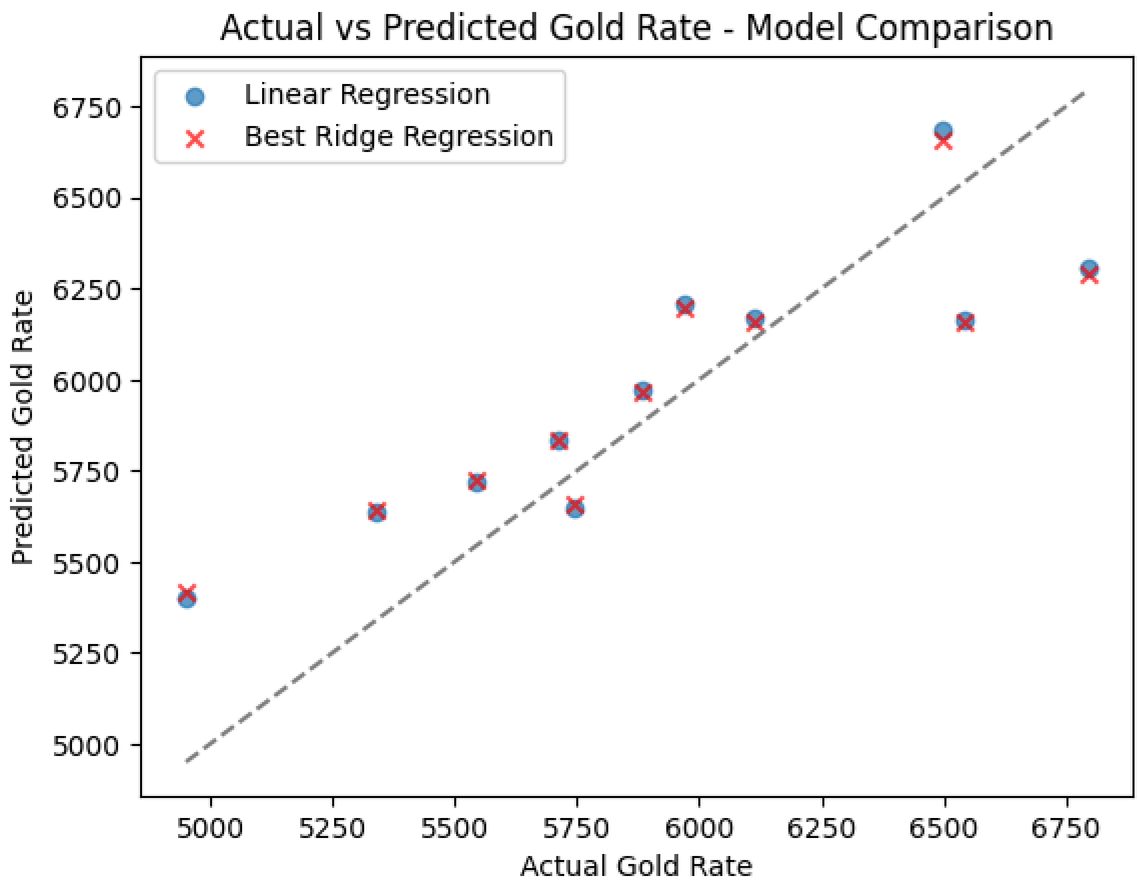
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hyperparameters: {'solver': 'saga', 'fit_intercept': True, 'alpha': 1.2
067926406393288}
```

In [ ]:
```
# Calculate the metrics for the best model
mse_best = mean_squared_error(y_test, y_pred_best)
rmse_best = np.sqrt(mse_best)
r2_best = r2_score(y_test, y_pred_best)

print(f"Best Model Mean Squared Error: {mse_best}")
print(f"Best Model Root Mean Squared Error: {rmse_best}")
print(f"Best Model R-squared: {r2_best}")
```

```
Best Model Mean Squared Error: 77648.51861180142
Best Model Root Mean Squared Error: 278.6548377685222
Best Model R-squared: 0.7173505784453535
```

In [ ]:
```
# comparing results of models through plots

import matplotlib.pyplot as plt

# Create a figure and an axes
fig, ax = plt.subplots()

# Plot the actual vs predicted values for the original linear regression moc
ax.scatter(y_test, y_pred, label='Linear Regression', alpha=0.7)

# Plot the actual vs predicted values for the best Ridge regression model
ax.scatter(y_test, y_pred_best, label='Best Ridge Regression', alpha=0.7, ma


# Add labels and title
ax.set_xlabel('Actual Gold Rate')
ax.set_ylabel('Predicted Gold Rate')
ax.set_title('Actual vs Predicted Gold Rate - Model Comparison')

# Add a legend
ax.legend()

# Add a diagonal line for reference
# Get the minimum and maximum values directly without indexing
min_value = min(y_test.min().values[0], y_pred.min(), y_pred_best.min())
```

```
max_value = max(y_test.max().values[0], y_pred.max(), y_pred_best.max())


ax.plot([min_value, max_value], [min_value, max_value], color='gray', linest

# Show the plot
plt.show()
```



2. Random Forest Regressor

In [ ]:
```
# using random forest regressor

# Initialize the RandomForestRegressor
rf_regressor = RandomForestRegressor(random_state=42)

# Define the parameter grid for RandomizedSearchCV (expanded for RandomFores
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2'],
}
```

In [ ]:
```
# Initialize RandomizedSearchCV for RandomForest
random_search_rf = RandomizedSearchCV(
    estimator=rf_regressor,
```

```
    param_distributions=param_grid_rf,
    n_iter=10,  # Number of random combinations to try
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    random_state=42,
    verbose=2
)
```

In [ ]:
```
# Fit the RandomForest model with RandomizedSearchCV
random_search_rf.fit(X_train_scaled, y_train.values.ravel()) # .ravel() to h

# Print the best hyperparameters for RandomForest
print("Best hyperparameters for RandomForest:", random_search_rf.best_params
```

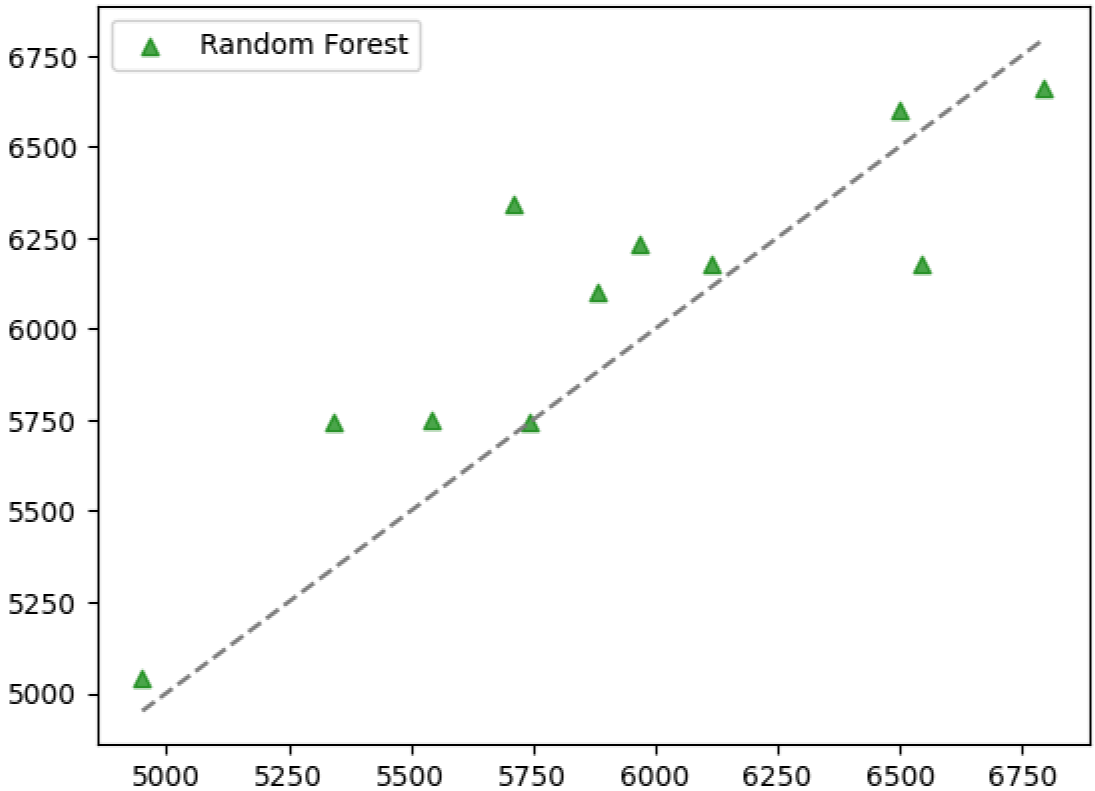
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best hyperparameters for RandomForest: {'n_estimators': 100, 'min_samples_sp
lit': 5, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': 20}
```

In [ ]:
```
# Get the best RandomForest model
best_rf = random_search_rf.best_estimator_

# Predict using the best RandomForest model
y_pred_rf = best_rf.predict(X_test_scaled)
```

In [ ]:
```
# Evaluate the best RandomForest model
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest - Mean Squared Error: {mse_rf}")
print(f"Random Forest - Root Mean Squared Error: {rmse_rf}")
print(f"Random Forest - R-squared: {r2_rf}")
```

```
Random Forest - Mean Squared Error: 81500.73831135004
Random Forest - Root Mean Squared Error: 285.4833415654056
Random Forest - R-squared: 0.703328061477292
```

In [ ]:
```
# ... (rest of your plotting code, modified to include Random Forest results
fig, ax = plt.subplots()

# Plot for Random Forest
ax.scatter(y_test, y_pred_rf, label='Random Forest', alpha=0.7, marker='^',

# Update legend and limits (as before)
ax.legend()

min_value = min(y_test.min().values[0], y_pred.min(), y_pred_best.min(), y_p
max_value = max(y_test.max().values[0], y_pred.max(), y_pred_best.max(), y_p
ax.plot([min_value, max_value], [min_value, max_value], color='gray', linest

plt.show()
```



3. ARIMA Model

In [ ]:
```
# Use arima model

# Prepare the data for ARIMA (using 'Goldrate' as the time series)
gold_prices = gold_usdinr_cleaned['Goldrate']

# Split data into training and testing sets
train_size = int(len(gold_prices) * 0.8)
train, test = gold_prices[0:train_size], gold_prices[train_size:len(gold_pri
```

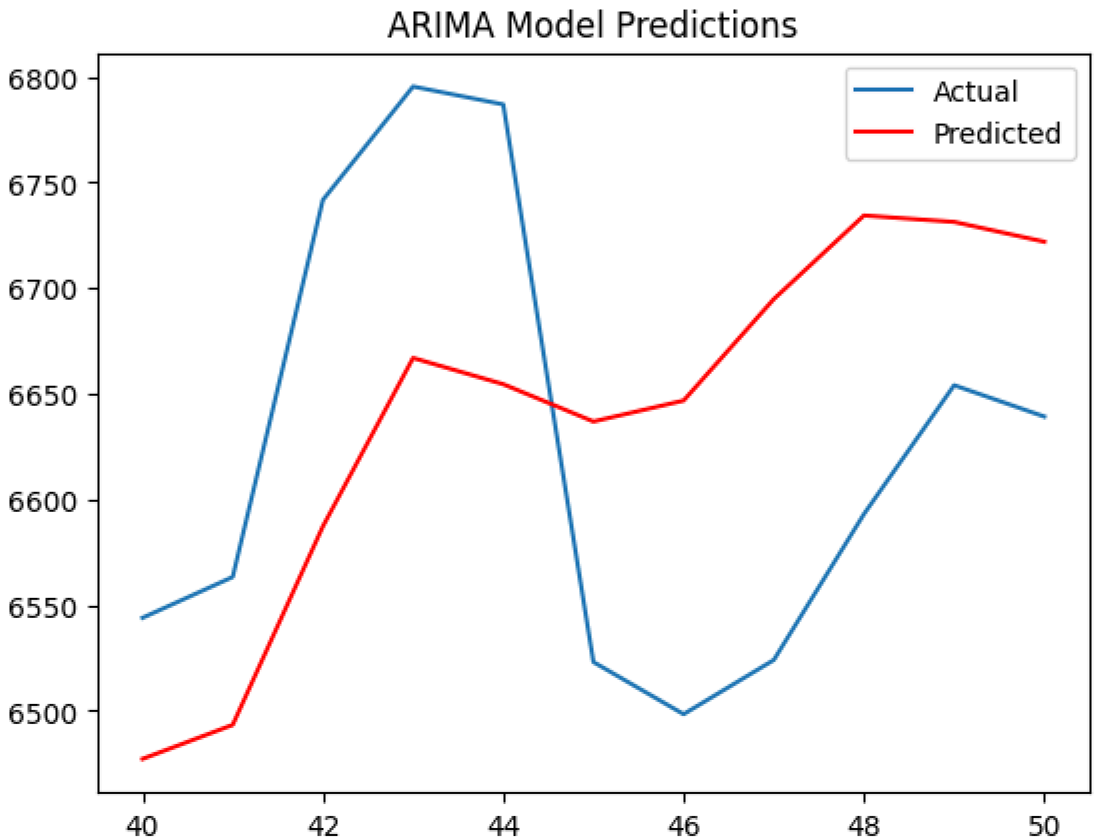
In [ ]:
```
# Fit the ARIMA model
# (p, d, q) are the order of the model. You might need to tune these paramet
model = ARIMA(train, order=(5, 1, 0))  # Example order, adjust as needed
model_fit = model.fit()
```

In [ ]:
```
# Make predictions
predictions = model_fit.predict(start=len(train), end=len(gold_prices)-1)

# Evaluate the model
rmse = np.sqrt(mean_squared_error(test, predictions))
print(f'Test RMSE: {rmse}')
```

```
Test RMSE: 122.12108130747129
```

In [ ]:
```
# Plot the results
plt.plot(test, label='Actual')
plt.plot(predictions, color='red', label='Predicted')
plt.legend()
```

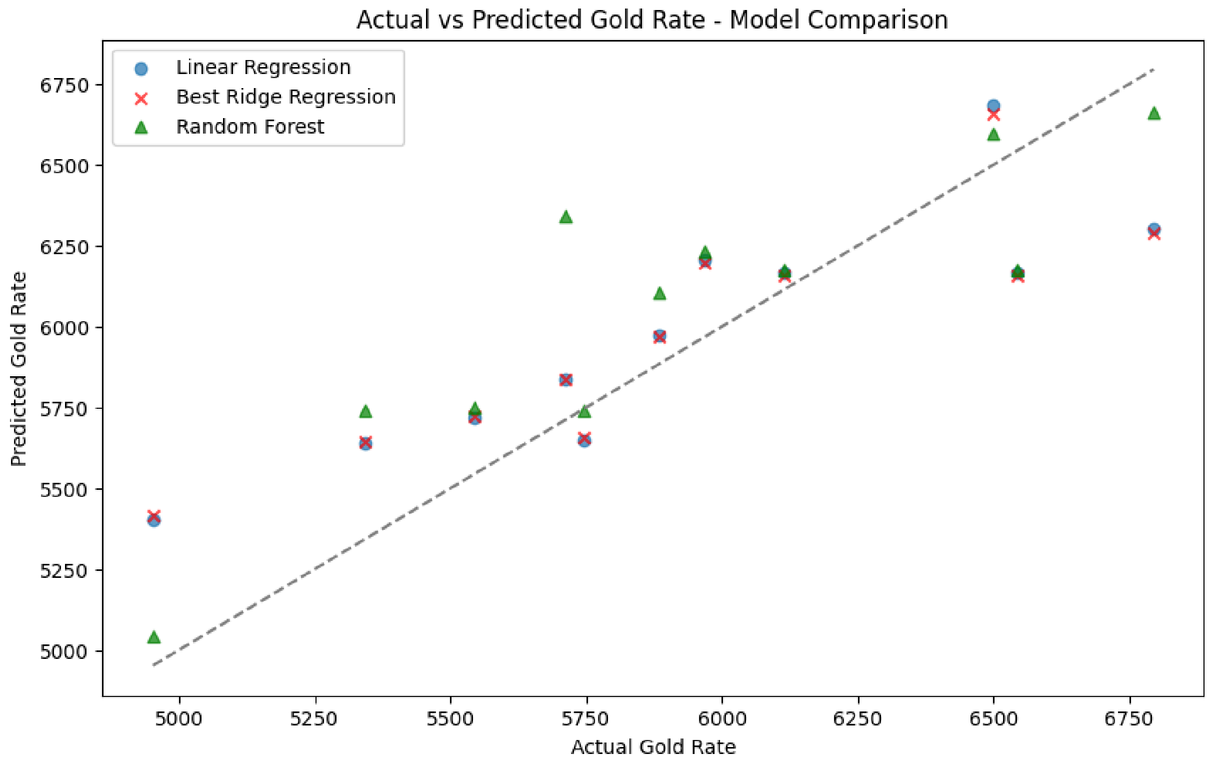
```
plt.title('ARIMA Model Predictions')
plt.show()
```



5. Results and Evaluation

```
In [ ]: # compare linear and random forest regression models

# Create a figure and an axes for the comparison plot
fig, ax = plt.subplots(figsize=(10, 6))

# Plot the actual vs predicted values for the original linear regression moc
ax.scatter(y_test, y_pred, label='Linear Regression', alpha=0.7)

# Plot the actual vs predicted values for the best Ridge regression model
ax.scatter(y_test, y_pred_best, label='Best Ridge Regression', alpha=0.7, ma

# Plot for Random Forest
ax.scatter(y_test, y_pred_rf, label='Random Forest', alpha=0.7, marker='^', 

# Add labels, title, legend, and diagonal line (as before)
ax.set_xlabel('Actual Gold Rate')
ax.set_ylabel('Predicted Gold Rate')
ax.set_title('Actual vs Predicted Gold Rate - Model Comparison')
ax.legend()

min_value = min(y_test.min().values[0], y_pred.min(), y_pred_best.min(), y_p
max_value = max(y_test.max().values[0], y_pred.max(), y_pred_best.max(), y_p
ax.plot([min_value, max_value], [min_value, max_value], color='gray', linest
```

```
plt.show()
```



In [ ]:

```
# Actual vs Predicted Data
y_test_values = range(len(y_test))  # X-axis (Sample Index)
y_actual = y_test.values

# Model Predictions
predictions = {
    "Linear Regression": y_pred_lr,
    "Best Ridge Regression": y_pred_ridge,
    "Random Forest": y_pred_rf,
}

# Plot
plt.figure(figsize=(12, 6))
plt.plot(y_test_values, y_actual, label="Actual", linestyle='dashed', color=

for model, y_pred in predictions.items():
    plt.plot(y_test_values, y_pred, label=model)

plt.legend()
plt.title("Gold Price Predictions vs Actual Values")
plt.xlabel("Test Sample Index")
plt.ylabel("Gold Price (INR)")
plt.show()
```

In [ ]:

```
# Create a DataFrame for easy comparison of metrics
results_df = pd.DataFrame({
    'Model': ['Linear Regression', 'Best Ridge Regression', 'Random Forest']
    'MSE': [mse, mse_best, mse_rf],
    'RMSE': [rmse, rmse_best, rmse_rf],
    'R-squared': [r2, r2_best, r2_rf]
})

results_df
```

Out[ ]:

	Model	MSE	RMSE	R-squared
0	Linear Regression	75693.831473	122.121081	0.724466
1	Best Ridge Regression	77648.518612	278.654838	0.717351
2	Random Forest	81500.738311	285.483342	0.703328

6. Deployment Using Gradio

In [211…

```
# save the models

joblib.dump(scaler, 'scaler.pkl')
joblib.dump(lin_reg, 'Regression_model.pkl')
joblib.dump(best_ridge, 'best_ridge_model.pkl')
joblib.dump(best_rf, 'best_random_forest_model.pkl')

print("Models saved successfully.")
```

```
Models saved successfully.
```

In [ ]:

```
# prompt: craeate a function to calculate goldrate

def predict_gold_rate(usd_inr_value):
```

```
    """
    Predicts the gold rate based on the USD/INR exchange rate using a pre-tr

    Args:
        usd_inr_value (float): The USD/INR exchange rate.

    Returns:
        float: The predicted gold rate.
    """
    try:
        # Load the pre-trained scaler and model
        scaler = joblib.load('scaler.pkl')
        model = joblib.load('/content/Regression_model.pkl')  # Use the best

        # Reshape the input for prediction
        usd_inr_scaled = scaler.transform(np.array([[usd_inr_value]]))

        # Make the prediction
        predicted_gold_rate = model.predict(usd_inr_scaled)

        return predicted_gold_rate[0][0]  # Extract the predicted value

    except FileNotFoundError:
        return "Error: Model files not found. Please ensure 'scaler.pkl' and
    except Exception as e:
        return f"An error occurred: {e}"
```

In [212…

```
# prompt: use gradio for predict gold rate

iface = gr.Interface(
    fn=predict_gold_rate,
    inputs=gr.Number(label="USD/INR Exchange Rate"),
    outputs= gr.Number(label="output"),
    title="Gold Rate Prediction🥇",
    description="Enter the USD/INR exchange rate to predict the gold rate.",
)

iface.launch(share=True)
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://1050d650b8d3516dcc.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgr
ades, run `gradio deploy` from the terminal in the working directory to depl
oy to Hugging Face Spaces (https://huggingface.co/spaces)
```

Gold Rate Prediction 🥇

Enter the USD/INR exchange rate to predict the gold rate.

USD/INR Exchange Rate

0

Clear

Submit

output

0

Flag

Built with Gradio · Settings

Out[212...

7.Key Observations

✅ Linear Regression performs the best with an R^2 score of 0.724 and the lowest MSE (122.12).

✅ Ridge Regression (a regularized version of Linear Regression) performs slightly worse, indicating that overfitting might not be a major issue.

✅ Random Forest performs the worst among the three models, suggesting that it may not capture the relationship between features effectively.

This notebook was converted with convert.ploomber.io