Load Data & Initial Inspection

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Define the path to the dataset
PATH = 'Dollar_Rial_Price_Dataset.csv'

# Load the dataset
df = pd.read_csv(PATH)

# Display the first few rows of the dataset
df.head()
```

Out[2]:

	Open Price	Low Price	High Price	Close Price	Change Amount	Change Percent	Gregorian Date	Persian Date
0	1073800	1073800	1089200	1088700	15700	1.46%	2025/11/01	1404/08/10
1	1071300	1071300	1076700	1073000	3900	0.36%	2025/10/30	1404/08/08
2	1069050	1066800	1074700	1069100	200	0.02%	2025/10/29	1404/08/07
3	1078550	1069300	1083200	1069300	10500	0.98%	2025/10/28	1404/08/06
4	1079150	1077300	1086200	1079800	1400	0.13%	2025/10/27	1404/08/05

In [3]:
```
df.shape
```

Out[3]: (3695, 8)

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3695 entries, 0 to 3694
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Open Price      3695 non-null   int64 
 1   Low Price       3695 non-null   int64 
 2   High Price      3695 non-null   int64 
 3   Close Price     3695 non-null   int64 
 4   Change Amount   3695 non-null   object
 5   Change Percent  3695 non-null   object
 6   Gregorian Date  3695 non-null   object
 7   Persian Date    3695 non-null   object
dtypes: int64(4), object(4)
memory usage: 231.1+ KB
```

In [5]:
```
df.describe().T.round(2).style.format('{:,.2f}')
```

Out[5]:

	count	mean	std	min	25%	50%	75%	n
Open Price	3,695.00	229,552.92	259,048.77	13,440.00	34,540.00	119,600.00	310,425.00	1,170,55(
Low Price	3,695.00	227,810.40	257,209.25	13,227.00	34,000.00	118,400.00	307,015.00	1,158,30(
High Price	3,695.00	231,449.59	261,386.43	13,440.00	34,750.00	121,000.00	314,470.00	1,180,70(
Close Price	3,695.00	229,683.76	259,364.62	13,350.00	34,295.00	119,450.00	310,028.00	1,178,60(

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Open Price        0
Low Price         0
High Price        0
Close Price       0
Change Amount     0
Change Percent    0
Gregorian Date    0
Persian Date      0
dtype: int64
```

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
np.int64(0)
```

In [8]:
```
# Convert all columns to lowercase and replace spaces with underscores
df.columns = df.columns.str.lower().str.replace(' ', '_')
df.columns.tolist()
```

Out[8]:
```
['open_price',
 'low_price',
 'high_price',
 'close_price',
 'change_amount',
 'change_percent',
 'gregorian_date',
 'persian_date']
```

In [9]:
```
# Drop persian_date column
df.drop(columns=['persian_date'], inplace=True)
```

In [10]:
```
# Convert Gregorian Date to datetime format
df['gregorian_date'] = pd.to_datetime(df['gregorian_date'], format='%Y/%m/%d')

# Sort the DataFrame by date and reset the index
df.sort_values('gregorian_date', inplace=True)
df.reset_index(drop=True, inplace=True)
```

```
# Set Gregorian Date as the index
df.set_index('gregorian_date', inplace=True)
df.index.min(), df.index.max()
```

Out[10]: (Timestamp('2011-11-26 00:00:00'), Timestamp('2025-11-01 00:00:00'))

In [11]:
```
df.head()
```

Out[11]:

	open_price	low_price	high_price	close_price	change_amount	change_per
gregorian_date						
2011-11-26	13700	13700	13700	13700	260	1.
2011-11-27	13440	13440	13440	13440	260	1.
2011-11-28	13495	13227	13667	13350	-	
2011-11-30	13580	13510	13830	13590	90	0.
2011-12-03	13638	13504	13833	13623	13	(

In [12]:
```
'''
    Recompute 'change_amount' & 'change_percentage' columns based due to data type
        - Both columns are read as object types, likely due to placeholder strings
        - Existing values are inconsistently calculated, leading to inaccuracies.
    Calculation formulas:
        - change_amount = current_price - previous_price
        - change_percentage = (change_amount / previous_price) * 100
'''
df['previous_close'] = df['close_price'].shift(1)
df['change_amount'] = df['close_price'] - df['previous_close']
df['change_percent'] = (df['change_amount'] / df['previous_close']) * 100
df.drop(columns=['previous_close'], inplace=True)
df.dropna(inplace=True)
df.head()
```

Out[12]:

	open_price	low_price	high_price	close_price	change_amount	change_per
gregorian_date						
2011-11-27	13440	13440	13440	13440	-260.0	-1.89
2011-11-28	13495	13227	13667	13350	-90.0	-0.66
2011-11-30	13580	13510	13830	13590	240.0	1.79
2011-12-03	13638	13504	13833	13623	33.0	0.24
2011-12-07	13494	13391	13658	13493	-130.0	-0.95

In [13]:
```
# Statistical summary after cleaning
df.describe().T.round(2).style.format('{:,.2f}')
```

Out[13]:

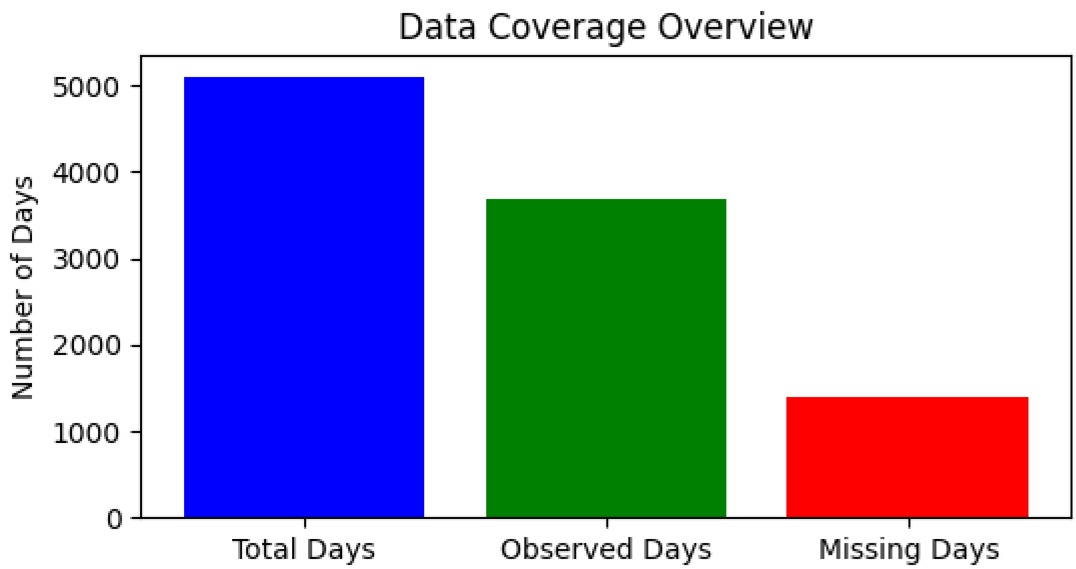
	count	mean	std	min	25%	50%	75
open_price	3,694.00	229,611.35	259,059.48	13,440.00	34,540.00	119,615.00	310,437.:
low_price	3,694.00	227,868.36	257,219.94	13,227.00	34,000.00	118,450.00	307,057.:
high_price	3,694.00	231,508.54	261,397.25	13,440.00	34,750.00	121,019.00	314,755.(
close_price	3,694.00	229,742.23	259,375.37	13,350.00	34,300.00	119,475.00	310,029.(
change_amount	3,694.00	291.01	6,256.53	-102,250.00	-333.00	10.00	800.(
change_percent	3,694.00	0.14	2.11	-16.32	-0.38	0.02	0.:

Exploratory Data Analysis (EDA)

In [14]:

```
# Check the data frequency and date range (expected: daily data with minimal missin

# Check date range and frequency
date_range = (df.index.min(), df.index.max())
total_days = (df.index.max() - df.index.min()).days
observed_days = df.shape[0]
missing_days = total_days - observed_days
frequency = 'Daily' if missing_days / total_days < 0.1 else 'Irregular'

# Print date range and frequency information
print(f"Date Range: {date_range[0].date()} to {date_range[1].date()}")
print(f"Total Days: {total_days}, Observed Days: {observed_days}, Missing Days: {mi
print(f"Data Frequency: {frequency}")

# Simple bar plot of total_days, observed_days, and missing_days
plt.figure(figsize=(6, 3))
plt.bar(['Total Days', 'Observed Days', 'Missing Days'], [total_days, observed_days
plt.title('Data Coverage Overview')
plt.ylabel('Number of Days')
plt.show()
```

```
Date Range: 2011-11-27 to 2025-11-01
Total Days: 5088, Observed Days: 3694, Missing Days: 1394
Data Frequency: Irregular
```



In [15]:
```python
# Find the missing dates and display them with their value counts
missing_dates = pd.date_range(start=date_range[0], end=date_range[1])
missing_dates = missing_dates.difference(df.index)
missing_days_of_week = missing_dates.to_series().dt.day_name().value_counts()
print(f"Missing Days of the Week: ({len(missing_dates)})\n{missing_days_of_week}")

## What percent of data is missing?
missing_percentage = (len(missing_dates) / total_days) * 100
print(f"\nPercentage of Missing Data: {missing_percentage:.2f}%")
```

```
Missing Days of the Week: (1395)
Friday       619
Thursday     437
Sunday        76
Saturday      73
Tuesday       71
Monday        66
Wednesday     53
Name: count, dtype: int64

Percentage of Missing Data: 27.42%
```

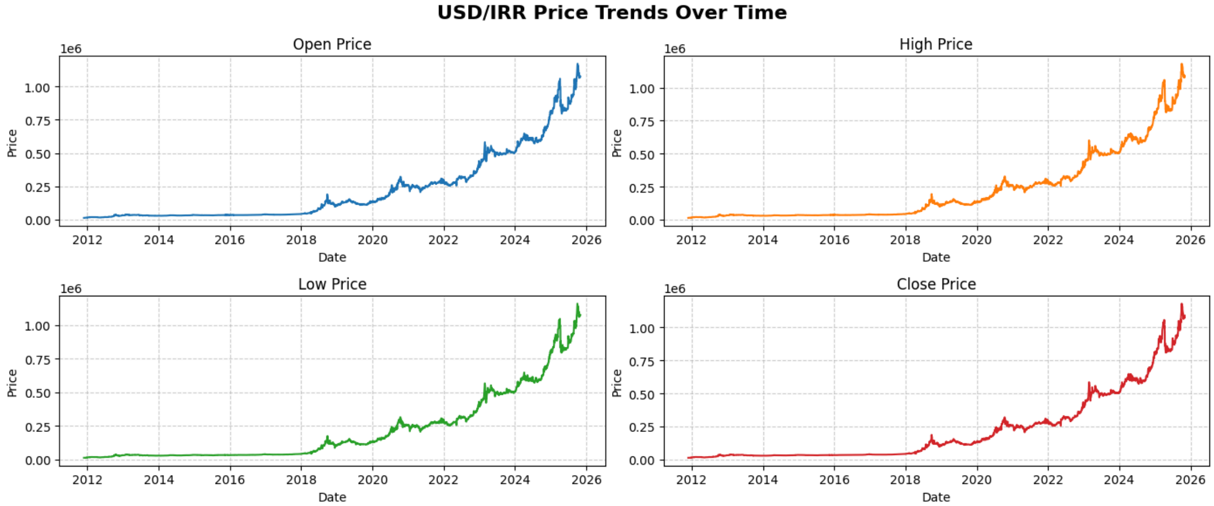
Handling Irregular Frequency - Reindex using Iran's business week (Sat–Thu), forward-fill missing holidays, and keep Fridays excluded. (NOT DOING IT HERE).

The reason why I am choosing to not treat the irregular frequency is because one I reindex using Iran's business week (Sat-Thu), and forward-fill the missing holidays, keeping Fridays excluded, this creates a hyper-persistent series via reindex + forward-fill. That would mean literally copying yesterday's close across holidays and closures, inflating lag-1 autocorrelation. This would mislead any time series model to think the series is more predictable than it actually is. Hence, the model will be overfitted and perform poorly out-of-sample, copying the previous day's close price, not really learning any patterns. Therefore, I will keep the irregular frequency as is, and let the model learn from the actual data without artificial inflation of autocorrelation. For that, I will not use ARIMA. We'll use models that can handle irregular frequency better, such as Prophet, LSTM, or Random Forests.

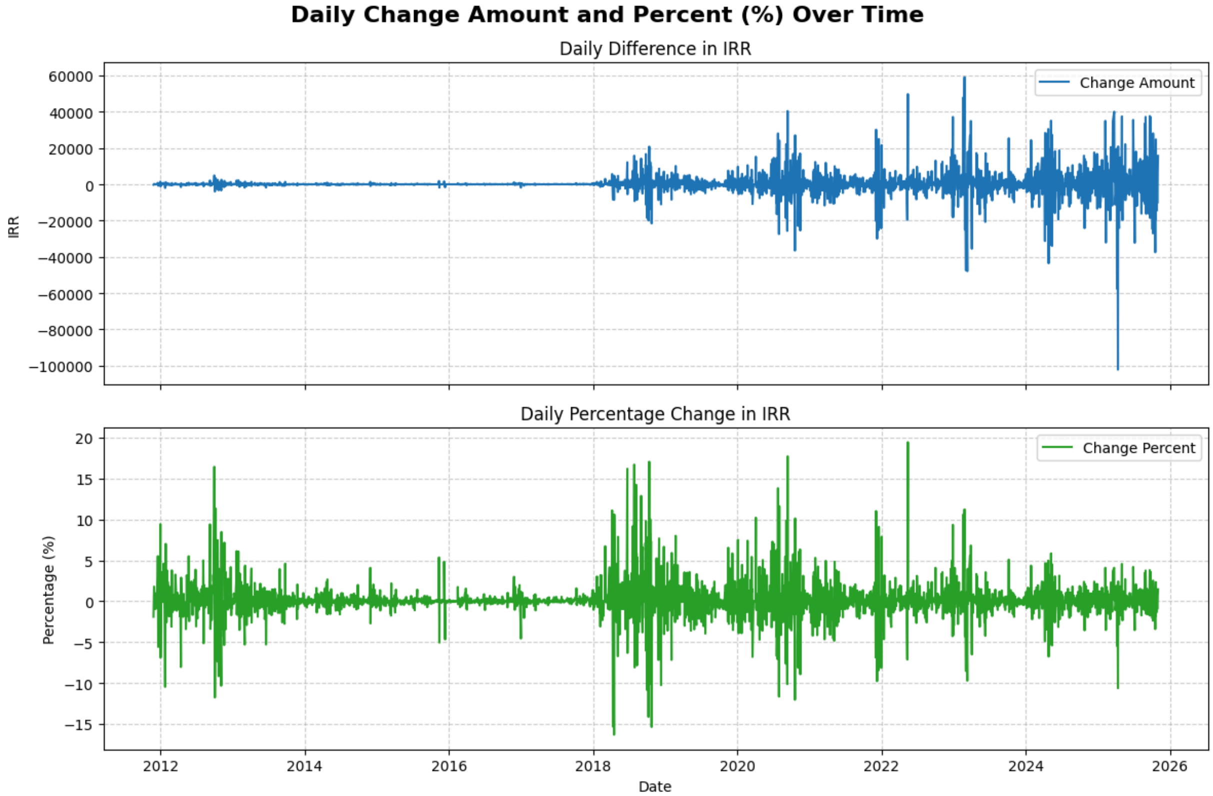
In [16]:
```
# Handling Irregular Frequency - Reindex using Iran's business week (Sat-Thu), forw
# iran_business_days = pd.bdate_range(start=date_range[0], end=date_range[1], freq=
# df = df.reindex(iran_business_days)
# df.ffill(inplace=True)  # Forward-fill to handle missing holidays
```

In [17]:
```
# Find the missing dates and display them with their value counts
# missing_dates = pd.date_range(start=date_range[0], end=date_range[1])
# missing_dates = missing_dates.difference(df.index)
# missing_days_of_week = missing_dates.to_series().dt.day_name().value_counts()
# print(f"Missing Days of the Week: ({len(missing_dates)})\n{missing_days_of_week}"

## What percent of data is missing?
# missing_percentage = (len(missing_dates) / total_days) * 100
# print(f"\nPercentage of Missing Data: {missing_percentage:.2f}%")
```

In [18]:
```
# Plot price trends over time

# Define the price columns
price_columns = ['open_price', 'high_price', 'low_price', 'close_price']

# Create subplots: 2 rows, 2 columns
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 6))
axes = axes.flatten()  # Flatten to easily iterate
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red']

# Loop through columns and axes
for i, col in enumerate(price_columns):
    axes[i].plot(df.index, df[col], color=colors[i])
    axes[i].set_title(f'{col.replace("_", " ").title()}')
    axes[i].set_xlabel('Date')
    axes[i].set_ylabel('Price')
    axes[i].grid(True, linestyle='--', alpha=0.6)

# Add one big title for the entire figure
fig.suptitle('USD/IRR Price Trends Over Time', fontsize=16, fontweight='bold')

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



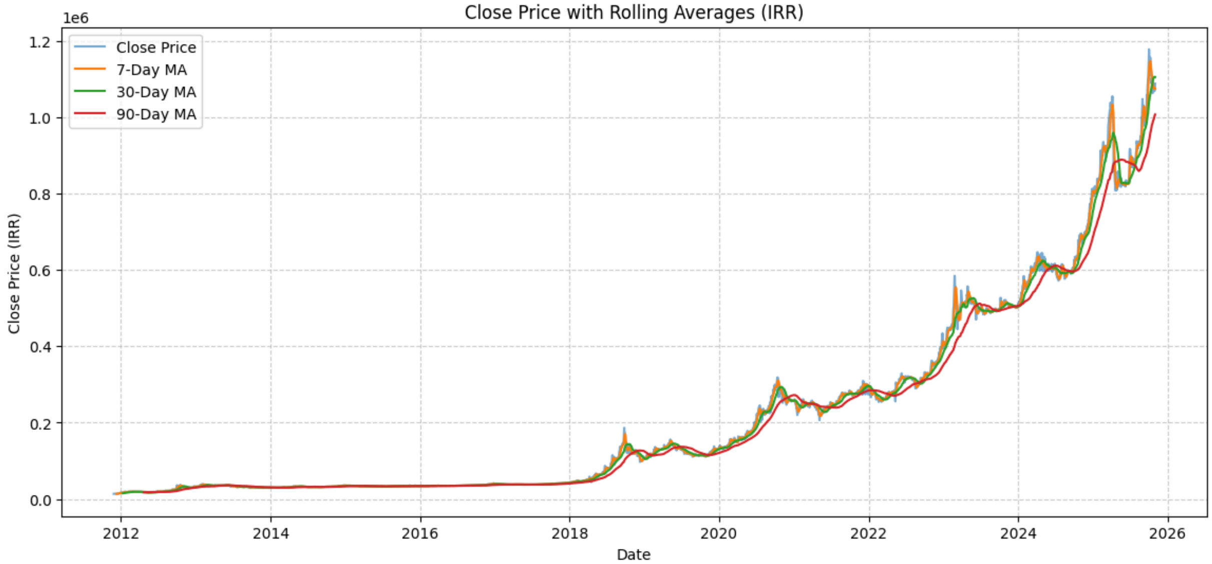
In [19]:
```
# Plot Change & Percent Amounts over time

# Create subplots: 2 rows, 1 column
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(12, 8), sharex=True)

# Plot change amount
axes[0].plot(df.index, df['change_amount'], color='tab:blue', label='Change Amount'
axes[0].set_title('Daily Difference in IRR') # Daily Change Amount
axes[0].set_ylabel('IRR')
axes[0].grid(True, linestyle='--', alpha=0.6)
axes[0].legend()

# Plot change percent
axes[1].plot(df.index, df['change_percent'], color='tab:green', label='Change Perce
axes[1].set_title('Daily Percentage Change in IRR') # Daily Change Percent
axes[1].set_ylabel('Percentage (%)')
axes[1].grid(True, linestyle='--', alpha=0.6)
axes[1].legend()

# Set the x-axis label for the bottom plot
axes[1].set_xlabel('Date')
fig.suptitle('Daily Change Amount and Percent (%) Over Time', fontsize=16, fontweig
plt.tight_layout()
plt.show()

# Print the min and max of change_amount and change_percent
print(f"Change Amount: Min = {df['change_amount'].min():,.2f}, Max = {df['change_am
print(f"Change Percent: Min = {df['change_percent'].min():.2f}%, Max = {df['change_
```




```
Change Amount: Min = -102,250.00, Max = 59,099.00
Change Percent: Min = -16.32%, Max = 19.45%
```

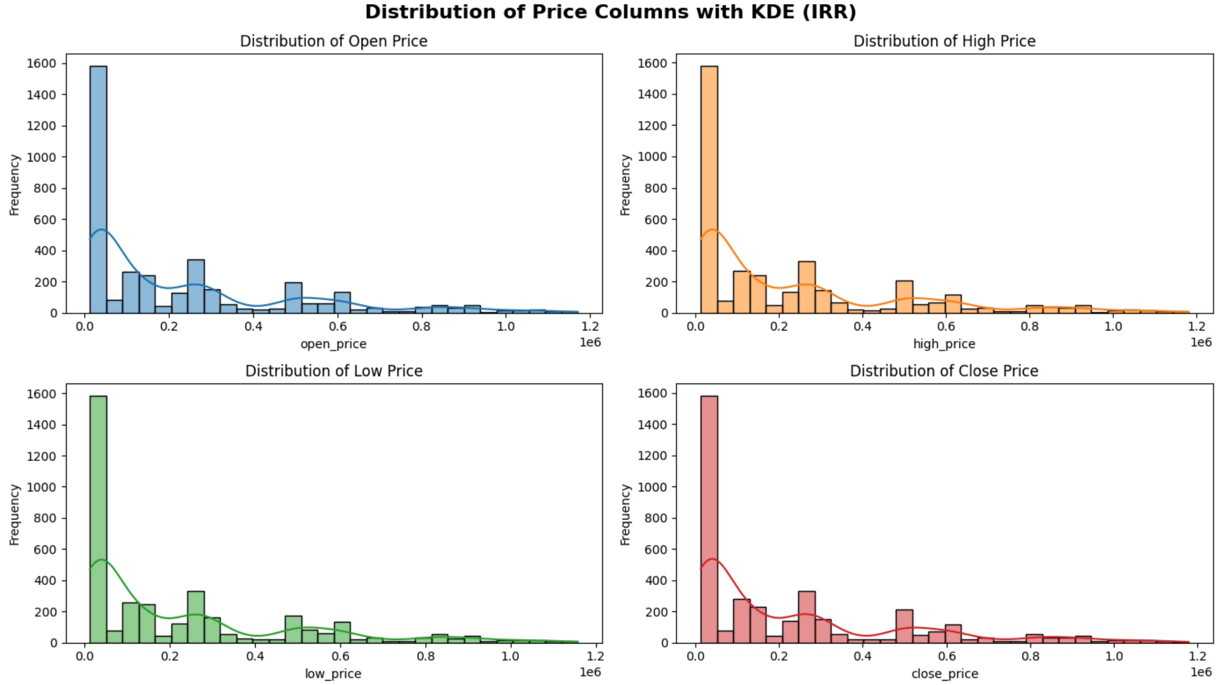
In [20]:

```
# Calculate rolling averages for 'close_price' (this will introduce NaN values at t

# Define rolling window sizes
rolling_windows = [7, 30, 90]

# Calculate rolling averages and add as new columns
for window in rolling_windows:
    df[f'close_price_ma_{window}'] = df['close_price'].rolling(window=window).mean(

# Plot close price with rolling averages
plt.figure(figsize=(14, 6))
plt.plot(df.index, df['close_price'], label='Close Price', alpha=0.6)
for window in rolling_windows:
    plt.plot(df.index, df[f'close_price_ma_{window}'], label=f'{window}-Day MA')
plt.title('Close Price with Rolling Averages (IRR)')
plt.xlabel('Date')
plt.ylabel('Close Price (IRR)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

# Drop the rolling average columns
df.drop(columns=['close_price_ma_7', 'close_price_ma_30', 'close_price_ma_90'], inp
```

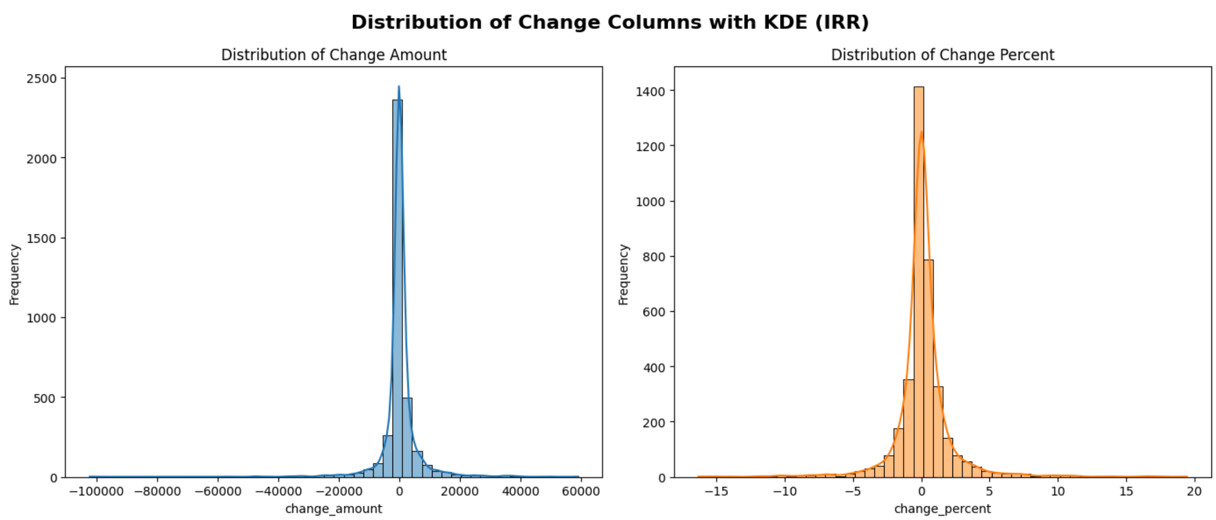


In [21]:

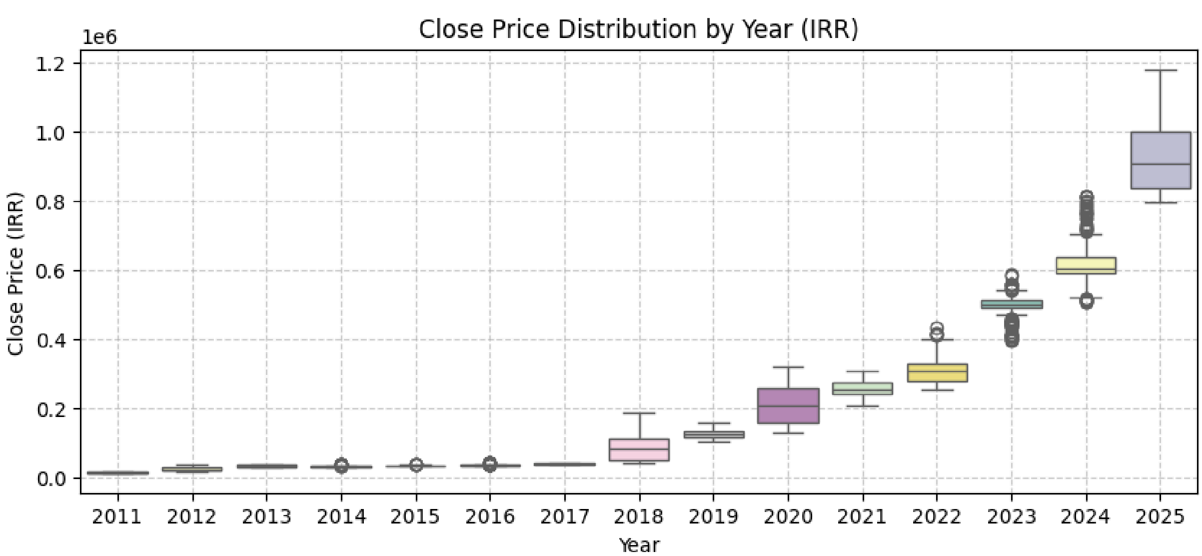
```
# Plot the distribution of price columns (histograms and KDE)

# 'price_columns' & 'colors' are already defined above

# Create subplots: 2 rows, 2 columns
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 8))
axes = axes.flatten()

for i, col in enumerate(price_columns):
    sns.histplot(df[col], bins=30, kde=True, color=colors[i], edgecolor='black', ax
    axes[i].set_title(f'Distribution of {col.replace("_", " ").title()}', fontsize=
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frequency')

plt.suptitle('Distribution of Price Columns with KDE (IRR)', fontsize=16, fontweigh
plt.tight_layout(rect=[0, 0, 1, 1])
plt.show()
```

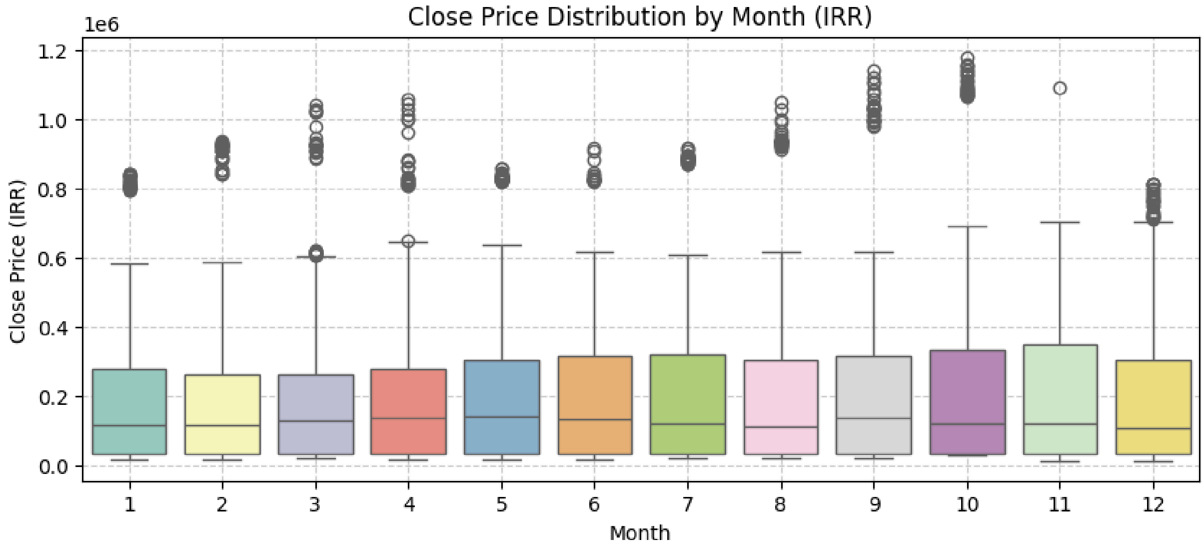


In [22]:
```
# Plot the distribution of change columns (histograms and KDE)

# 'colors' is already defined above

# Create 2 plots side by side
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

for i, col in enumerate(['change_amount', 'change_percent']):
    sns.histplot(df[col], bins=50, kde=True, color=colors[i], edgecolor='black', ax
    axes[i].set_title(f'Distribution of {col.replace("_", " ").title()}', fontsize=
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frequency')

plt.suptitle('Distribution of Change Columns with KDE (IRR)', fontsize=16, fontweig
plt.tight_layout(rect=[0, 0, 1, 1])
plt.show()
```



In [23]:
```
# Plot close price distribution by year and month

# Extract year and month from the index
df['year'] = df.index.year
df['month'] = df.index.month

# Boxplot of close price by year
plt.figure(figsize=(10, 4))
sns.boxplot(x='year', y='close_price', data=df, palette='Set3')
plt.title('Close Price Distribution by Year (IRR)')
plt.xlabel('Year')
plt.ylabel('Close Price (IRR)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

# Boxplot of close price by month
plt.figure(figsize=(10, 4))
sns.boxplot(x='month', y='close_price', data=df, palette='Set3')
plt.title('Close Price Distribution by Month (IRR)')
plt.xlabel('Month')
plt.ylabel('Close Price (IRR)')
plt.grid(True, linestyle='--', alpha=0.6)

plt.show()

# Print statistical summary by year and month
yearly_summary = df.groupby('year')['close_price'].describe().round(2)
monthly_summary = df.groupby('month')['close_price'].describe().round(2)

# Print in a pretty format
print("Yearly Close Price Summary (IRR):")
display(yearly_summary)
print("\nMonthly Close Price Summary (IRR):")
display(monthly_summary)

# Drop year and month columns as they were only needed for grouping
df.drop(columns=['year', 'month'], inplace=True)
```





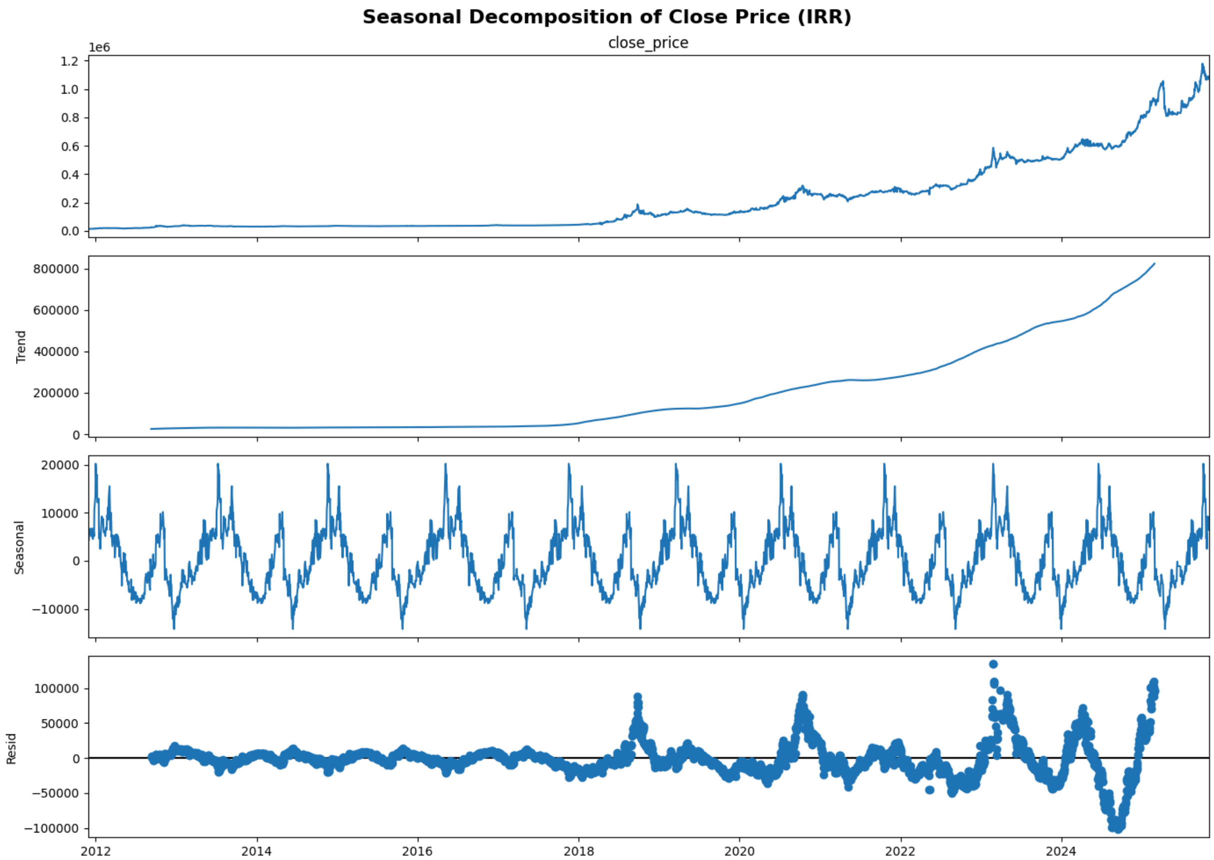
Yearly Close Price Summary (IRR):

year	count	mean	std	min	25%	50%	75%	max
2011	21.0	14315.62	832.48	13350.0	13590.0	14080.0	15150.00	15900.0
2012	245.0	22833.07	6005.72	15750.0	18500.0	19430.0	29050.00	36100.0
2013	234.0	33068.00	2613.25	29150.0	30412.5	32900.0	35387.50	38830.0
2014	284.0	31737.92	1400.27	29340.0	30877.5	31745.0	32520.00	35590.0
2015	249.0	33671.98	647.72	32140.0	33190.0	33836.0	33981.00	35750.0
2016	239.0	35388.85	1513.53	33800.0	34505.0	34970.0	35749.50	41210.0
2017	238.0	38820.84	1480.41	37240.0	37650.0	38210.0	39757.25	42880.0
2018	274.0	86743.39	34731.73	43283.0	51022.5	81520.0	114142.50	186680.0
2019	283.0	124892.92	10799.99	104500.0	115075.0	124020.0	131955.00	156010.0
2020	286.0	207748.73	54152.06	129500.0	157002.5	208210.0	257755.00	318526.0
2021	284.0	258870.22	21572.67	206480.0	242287.5	253720.0	275817.50	309710.0
2022	268.0	309168.25	37573.02	254170.0	277872.5	308730.0	327985.00	434360.0
2023	274.0	496442.77	29630.80	394830.0	490882.5	500825.0	511080.00	584930.0
2024	287.0	620827.86	60247.57	506430.0	590200.0	604850.0	637700.00	813200.0
2025	228.0	924703.07	99353.69	793450.0	834650.0	907200.0	998387.50	1178600.0

Monthly Close Price Summary (IRR):

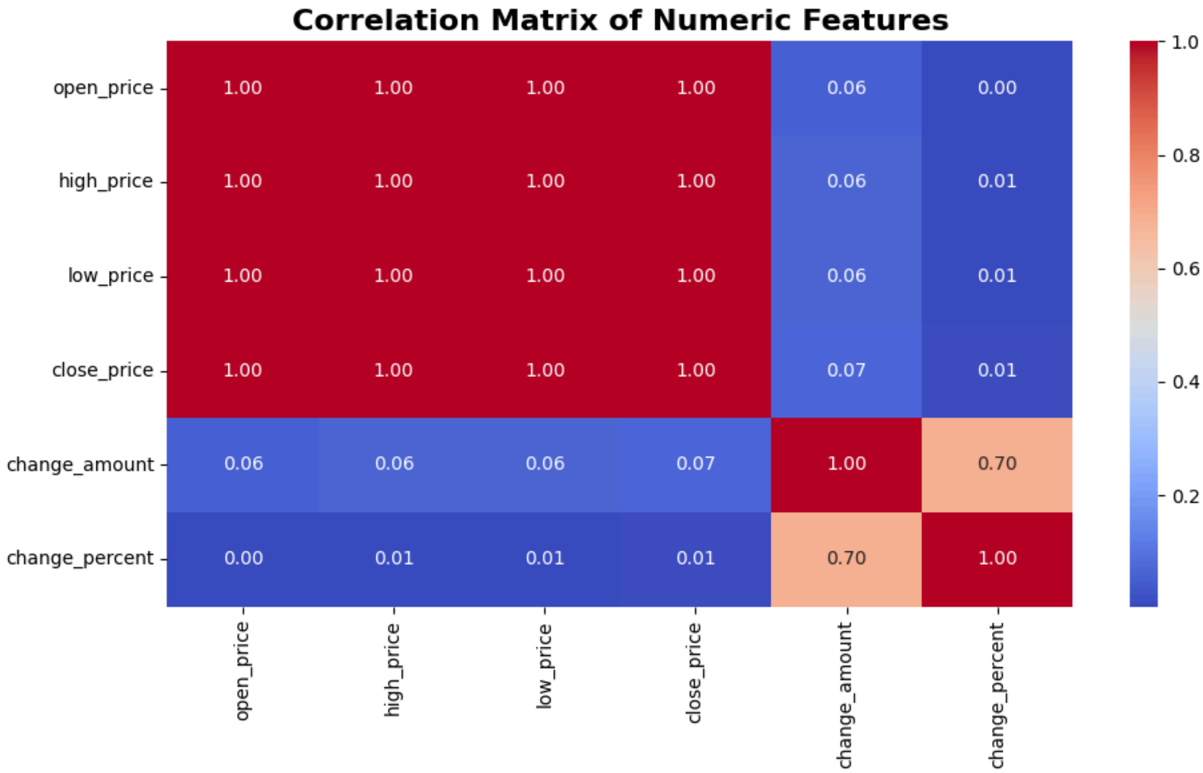
	count	mean	std	min	25%	50%	75%	max
month								
1	325.0	208615.77	234920.85	15800.0	34600.00	114500.0	278820.00	839300.0
2	288.0	215819.57	260912.93	17830.0	34498.00	116800.0	261475.00	935800.0
3	272.0	224395.77	270773.73	18520.0	34262.25	128860.0	262575.00	1038950.0
4	304.0	230254.14	266280.57	17100.0	34630.00	136165.0	278772.50	1055550.0
5	330.0	229446.65	255622.89	15750.0	34542.50	141416.0	304924.25	858700.0
6	279.0	215106.77	234062.63	17600.0	34515.00	131350.0	316825.00	917200.0
7	319.0	233893.74	256474.56	18990.0	33107.00	120240.0	319062.00	916000.0
8	321.0	241271.41	270042.44	20190.0	33558.00	113010.0	303110.00	1048300.0
9	309.0	256039.30	287139.69	21590.0	33930.00	137600.0	315560.00	1141300.0
10	327.0	274839.33	312038.04	29950.0	34600.00	121900.0	331115.00	1178600.0
11	299.0	213970.88	217212.31	13350.0	33940.00	120300.0	348607.00	1088700.0
12	321.0	208474.84	224193.55	13450.0	33970.00	108190.0	303380.00	813200.0

In [24]:
```
from statsmodels.tsa.seasonal import seasonal_decompose             # Seasonal deco

# Plot seasonal decomposition of close price
decomposition = seasonal_decompose(df['close_price'], model='additive', period=365)

fig = decomposition.plot()
fig.set_size_inches(14, 10)
plt.suptitle('Seasonal Decomposition of Close Price (IRR)', fontsize=16, fontweight
plt.tight_layout(rect=[0, 0, 1, 1])
plt.show()

'''
    In additive seasonal decomposition, the model computes: --> 'Close Price' = Tre
    - Trend: The long-term progression of the series (overall increase in close pri
    - Seasonal: The repeating short-term cycle (monthly patterns in close price).
    - Residual: The random noise left after removing trend and seasonal components.

    Residual --> Residual = Observed – (Trend + Seasonal)
'''
```



Out[24]: "\n   In additive seasonal decomposition, the model computes: --> 'Close Price' = Trend + Seasonal + Residual\n   - Trend: The long-term progression of the series (overall increase in close price over years).\n   - Seasonal: The repeating short-term cycle (monthly patterns in close price).\n   - Residual: The random noise left after removing trend and seasonal components.\n\n   Residual --> Residual = Observed - (Trend + Seasonal)\n"

In [25]:
```
# Compute Correlation Matrix (Pearson correlations)
corr_matrix = df[['open_price', 'high_price', 'low_price', 'close_price', 'change_a

# Plot the correlation matrix heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix of Numeric Features', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```



In [26]:
```
# Drop open_price, high_price, low_price as they are highly correlated with close_p
df.drop(columns=['open_price', 'high_price', 'low_price'], inplace=True)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3694 entries, 2011-11-27 to 2025-11-01
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   close_price     3694 non-null   int64  
 1   change_amount   3694 non-null   float64
 2   change_percent  3694 non-null   float64
dtypes: float64(2), int64(1)
memory usage: 244.5 KB
```

In [ ]: