Libraries

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Reading File

In [3]:
```
file_path = "E:\VamStar\AA.csv"
data = pd.read_csv(file_path)
data.head()
```

Out[3]:

	contract_id	published_date	start_date	duration_extension	outcome	second_place_out
0	1	5/16/2013	5/24/2013	10	won	
1	2	4/29/2013	6/21/2013	0	won	
2	3	6/9/2013	8/14/2014	6	won	
3	4	5/4/2013	5/17/2013	19	won	
4	5	12/18/2013	12/17/2013	6	won	

5 rows × 26 columns

Data Understanding

In [6]:
```
 # data types
print("Original data types:")
print(data.dtypes)
```

```
Original data types:
contract_id                int64
published_date            object
start_date                object
duration_extension         int64
outcome                   object
second_place_outcome      object
buyer                     object
region                    object
atc                       object
duration                   int64
contract_type             object
sku                       object
end_date_extension        object
participants_no            int64
quantity_annual            int64
quantity_total           float64
maximum_price_allowed    float64
active_ingredient         object
pack_strength             object
participants              object
participants_price        object
published_date_month      object
winner                    object
winner_price             float64
second_place              object
second_place_price       float64
dtype: object
```

In [7]:
```
print("Initial data info:")
print(data['winner_price'].describe())
```

```
Initial data info:
count    31.000000
mean      0.016300
std       0.011421
min       0.000010
25%       0.012250
50%       0.016990
75%       0.020000
max       0.050000
Name: winner_price, dtype: float64
```

Missing Values

In [4]:
```
print("Missing values count:", data['winner_price'].isna().sum())
```

```
Missing values count: 0
```

Duplicate Values

In [9]:
```
duplicate_count = data.duplicated().sum()
print("Duplicate values count:", duplicate_count)
```

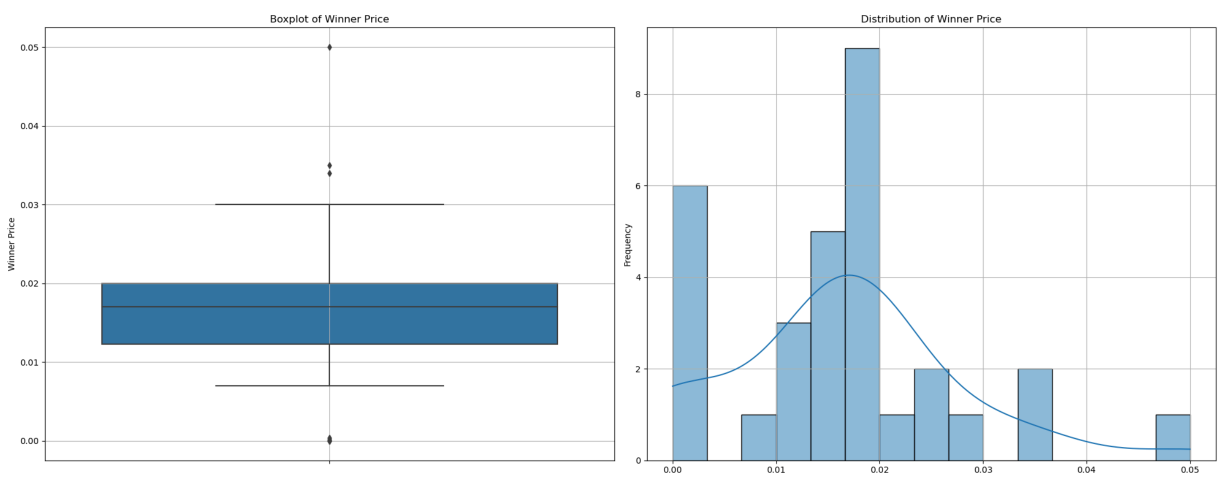
```
Duplicate values count: 0
```

EDA Techniques

Finding Outlier

Using BoxPlot

In [10]:
```
plt.figure(figsize=(20, 8))  # Increased size to make each subplot

# First subplot: Boxplot
plt.subplot(1, 2, 1)
sns.boxplot(y=data['winner_price'])
plt.title('Boxplot of Winner Price')
plt.ylabel('Winner Price')
plt.grid(True)

# Second subplot: Histogram with KDE
plt.subplot(1, 2, 2)
sns.histplot(data['winner_price'], kde=True, bins=15)
plt.title('Distribution of Winner Price')
plt.xlabel('Winner Price')
plt.ylabel('Frequency')
plt.grid(True)

# Show the entire figure with both plots
plt.tight_layout()
plt.show()
```



Using IQR - Trimming

Outliers can sometimes be removed without concern

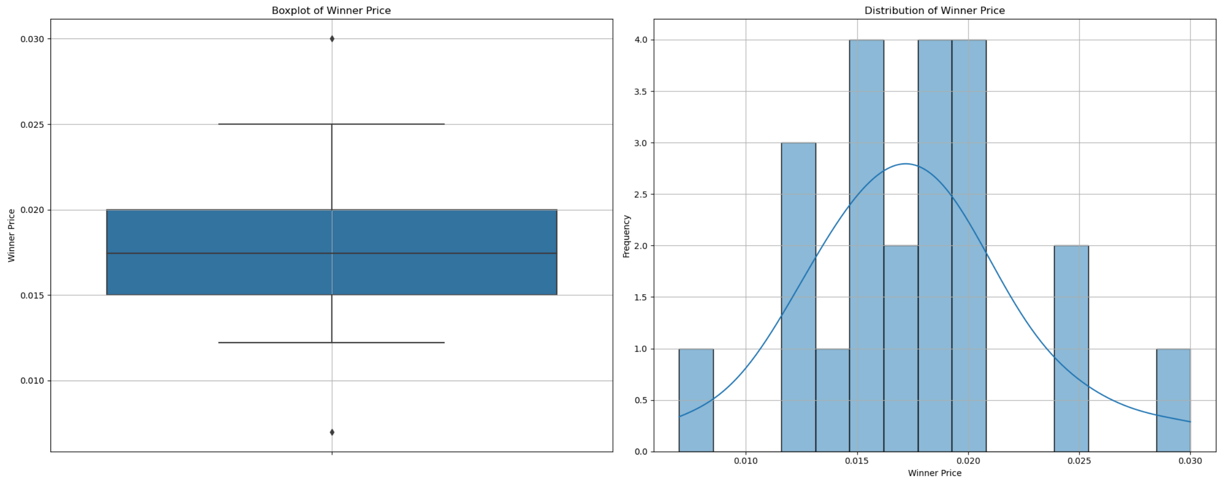
In [15]:
```
# Calculate Q1, Q3, and IQR
Q1 = data['winner_price'].quantile(0.25)
Q3 = data['winner_price'].quantile(0.75)
IQR = Q3 - Q1

# Determine outliers using IQR
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

In [14]:
```
# Round the lower and upper bounds to 5 decimal places
lower_bound_rounded = round(lower_bound, 5)
upper_bound_rounded = round(upper_bound, 5)

# Print the rounded lower and upper bounds
print("Lower Bound:", lower_bound_rounded)
print("Upper Bound:", upper_bound_rounded)
```

```
Lower Bound: 0.00063
Upper Bound: 0.03162
```

In [18]:
```
# Filter out the outliers to keep only non-outliers
data_clean_Trimming = data[(data['winner_price'] >= lower_bound) & (data['winner_pr
```

In [19]:
```
plt.figure(figsize=(20, 8))

# First subplot: Boxplot
plt.subplot(1, 2, 1)
sns.boxplot(y=data_clean_Trimming['winner_price'])
plt.title('Boxplot of Winner Price')
plt.ylabel('Winner Price')
plt.grid(True)

# Second subplot: Histogram with KDE
plt.subplot(1, 2, 2)
sns.histplot(data_clean_Trimming['winner_price'], kde=True, bins=15)
plt.title('Distribution of Winner Price')
plt.xlabel('Winner Price')
plt.ylabel('Frequency')
plt.grid(True)

# Show the entire figure with both plots
plt.tight_layout()  # Adjusts subplots to give some padding and prevent overlap
plt.show()
```

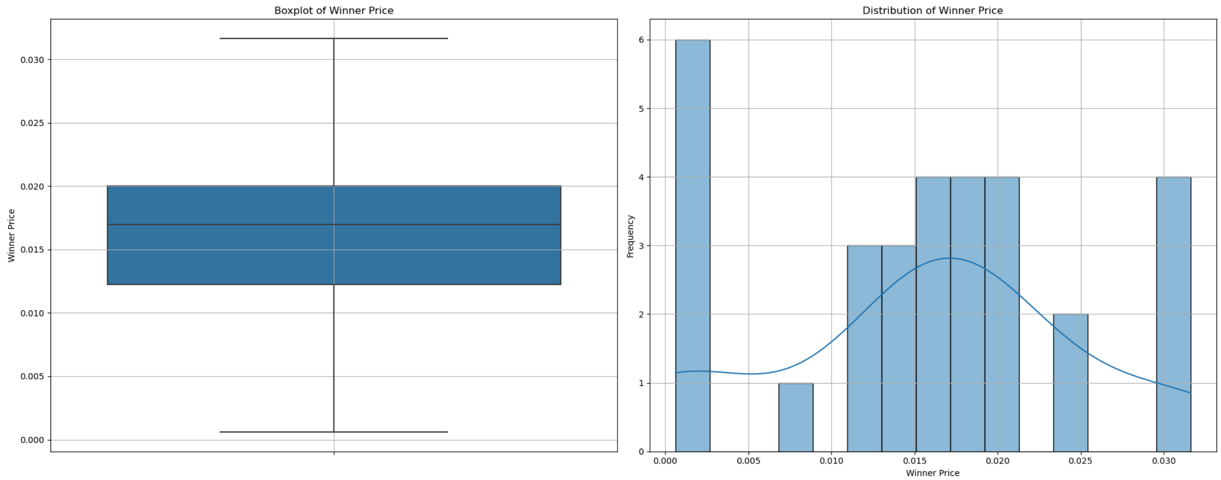


Using IQR - Capping

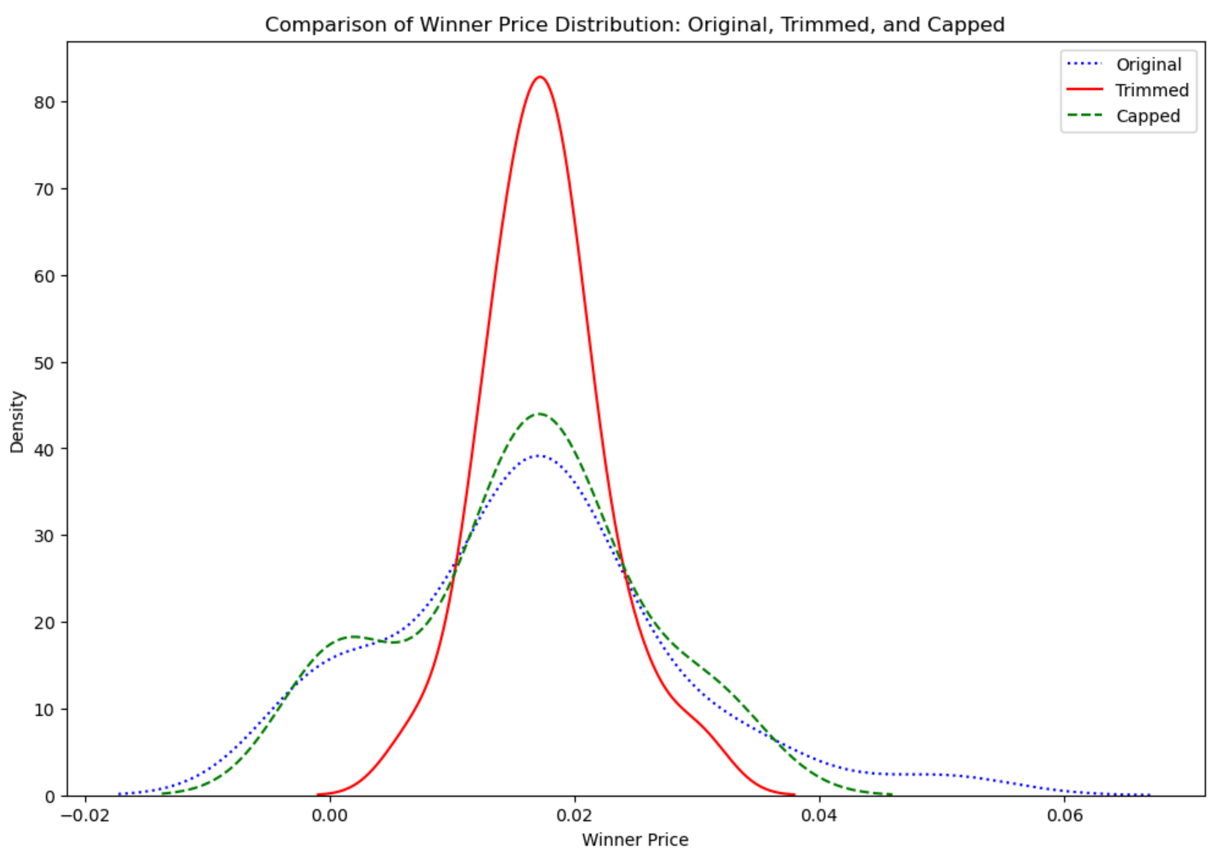
Sometimes deleting data due to outliers can lead to the loss of other useful features, so using the capping method may be a preferable alternative.

In [21]:
```
# Capping outliers
data_clean_Capping = data['winner_price'].apply(
    lambda x: max(min(x, upper_bound), lower_bound)
)
```

In [23]:
```
plt.figure(figsize=(20, 8))

# First subplot: Boxplot
plt.subplot(1, 2, 1)
sns.boxplot(y=data_clean_Capping)
plt.title('Boxplot of Winner Price')
plt.ylabel('Winner Price')
plt.grid(True)

# Second subplot: Histogram with KDE
plt.subplot(1, 2, 2)
sns.histplot(data_clean_Capping, kde=True, bins=15)
plt.title('Distribution of Winner Price')
plt.xlabel('Winner Price')
plt.ylabel('Frequency')
plt.grid(True)

# Show the entire figure with both plots
plt.tight_layout()
plt.show()
```



In [32]:
```
plt.figure(figsize=(12, 8))
sns.kdeplot(data['winner_price'], label='Original', color="blue", linestyle=':')
sns.kdeplot(data_clean_Trimming['winner_price'], label='Trimmed', color="red")
sns.kdeplot(data['capped_winner_price'], label='Capped', color="green", linestyle='

plt.title('Comparison of Winner Price Distribution: Original, Trimmed, and Capped')
plt.xlabel('Winner Price')
plt.ylabel('Density')
plt.legend()
plt.show()
```



The comparison between the original, trimmed, and capped distributions of winner_price illustrates the effects of each outlier handling method. Trimming removes outliers, resulting in a more compact and possibly normal distribution, ideal for analyses sensitive to extreme values. In contrast, capping truncates the distribution's tails and introduces peaks at the bounds, maintaining data integrity but potentially affecting natural distribution interpretations. Each method suits different analytical needs; trimming is useful for reducing data variability, while capping preserves dataset size, beneficial when data loss is a concern.