In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.read_csv('youtube_channel_real_performance_analytics.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	ID	Video Duration	Video Publish Time	Days Since Publish	Day	Month	Year	Day of Week	Revenue per 1000 Views (USD)	Monetized Playbacks (Estimate)	...	Watch (N Skipp (
0	0	201.0	2016-06-02 00:00:00	0	2	6	2016	Thursday	0.024	723.0	...	
1	1	391.0	2016-06-10 00:00:00	8	10	6	2016	Friday	0.056	727.0	...	
2	2	133.0	2016-06-14 00:00:00	4	14	6	2016	Tuesday	0.014	76.0	...	
3	3	14.0	2016-06-29 00:00:00	15	29	6	2016	Wednesday	0.004	18.0	...	
4	4	45.0	2016-07-01 00:00:00	2	1	7	2016	Friday	0.000	0.0	...	

5 rows × 70 columns

In [4]:
```
1 df.columns
```

Out[4]:
```
Index(['ID', 'Video Duration', 'Video Publish Time', 'Days Since Publis
h',
       'Day', 'Month', 'Year', 'Day of Week', 'Revenue per 1000 Views
(USD)',
       'Monetized Playbacks (Estimate)', 'Playback-Based CPM (USD)',
       'CPM (USD)', 'Ad Impressions', 'Estimated AdSense Revenue (US
D)',
       'DoubleClick Revenue (USD)', 'YouTube Ads Revenue (USD)',
       'Watch Page Ads Revenue (USD)', 'YouTube Premium (USD)',
       'Transaction Revenue (USD)', 'Transactions',
       'Revenue from Transactions (USD)', 'Reactions', 'Chat Messages C
ount',
       'Reminders Set', 'Stream Hours', 'Remix Views', 'Remix Count',
       'Subscribers from Posts', 'New Comments', 'Shares', 'Like Rate
(%)',
       'Dislikes', 'Likes', 'Unsubscribes', 'New Subscribers',
       'Returned Items (USD)', 'Unconfirmed Commissions (USD)',
       'Approved Commissions (USD)', 'Orders', 'Total Sales Volume (US
D)',
       'End Screen Click-Through Rate (%)', 'End Screen Impressions',
       'End Screen Clicks', 'Teaser Click-Through Rate (%)',
       'Teaser Impressions', 'Teaser Clicks', 'Card Click-Through Rate
(%)',
       'Card Impressions', 'Card Clicks', 'Views per Playlist Start',
       'Playlist Views', 'Playlist Watch Time (hours)',
       'Clip Watch Time (hours)', 'Clip Views',
       'YouTube Premium Watch Time (hours)', 'YouTube Premium Views',
       'Returning Viewers', 'New Viewers', 'Average Views per User',
       'Unique Viewers', 'Watched (Not Skipped) (%)', 'Feed Impression
s',
       'Average View Percentage (%)', 'Average View Duration', 'Views',
       'Watch Time (hours)', 'Subscribers', 'Estimated Revenue (USD)',
       'Impressions', 'Video Thumbnail CTR (%)'],
      dtype='object')
```

In [5]:
```
1 data = df[['Estimated Revenue (USD)','Day of Week','Video Publish Ti
```

In [6]:
```
1 data.shape
```

Out[6]:
```
(364, 9)
```

In [7]:
```
1 data.head()
```

Out[7]:

	Estimated Revenue (USD)	Day of Week	Video Publish Time	Views	Shares	Video Duration	Dislikes	Likes	Subscribers
0	0.561	Thursday	2016-06-02 00:00:00	23531.0	12.0	201.0	30.0	924.0	51.0
1	0.648	Friday	2016-06-10 00:00:00	11478.0	5.0	391.0	18.0	322.0	33.0
2	0.089	Tuesday	2016-06-14 00:00:00	6153.0	4.0	133.0	20.0	239.0	8.0
3	0.017	Wednesday	2016-06-29 00:00:00	4398.0	7.0	14.0	14.0	220.0	2.0
4	0.000	Friday	2016-07-01 00:00:00	14659.0	7.0	45.0	180.0	602.0	28.0

Data Cleaning & EDA

In [8]:
```
1 #Missing value
2 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364 entries, 0 to 363
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Estimated Revenue (USD)  364 non-null    float64
 1   Day of Week             364 non-null    object 
 2   Video Publish Time      364 non-null    object 
 3   Views                   364 non-null    float64
 4   Shares                  364 non-null    float64
 5   Video Duration          364 non-null    float64
 6   Dislikes                364 non-null    float64
 7   Likes                   364 non-null    float64
 8   Subscribers             364 non-null    float64
dtypes: float64(7), object(2)
memory usage: 25.7+ KB
```

In [9]:
```
1 #Duplicate Value
2 data[data.duplicated()]
```

Out[9]:

	Estimated Revenue (USD)	Day of Week	Video Publish Time	Views	Shares	Video Duration	Dislikes	Likes	Subscribers

In [10]:

```
data['Video Publish Time'] = pd.to_datetime(data['Video Publish Time
data['Publish Hour'] = data['Video Publish Time'].dt.hour
data['Publish Day'] = data['Video Publish Time'].dt.day
data['Publish Month'] = data['Video Publish Time'].dt.month
data['Publish Year'] = data['Video Publish Time'].dt.year
data.drop(columns=["Video Publish Time"], inplace=True)
```

```
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/36750983
7.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (ht
tps://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy)
  data['Video Publish Time'] = pd.to_datetime(data['Video Publish Tim
e'])
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/36750983
7.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (ht
tps://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy)
  data['Publish Hour'] = data['Video Publish Time'].dt.hour
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/36750983
7.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (ht
tps://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy)
  data['Publish Day'] = data['Video Publish Time'].dt.day
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/36750983
7.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (ht
tps://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy)
  data['Publish Month'] = data['Video Publish Time'].dt.month
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/36750983
7.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (ht
tps://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy)
  data['Publish Year'] = data['Video Publish Time'].dt.year
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/36750983
7.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (ht
```

```
tps://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data.drop(columns=["Video Publish Time"], inplace=True)
```

In [11]:
```
day_mapping = {
    "Monday": 1, "Tuesday": 2, "Wednesday": 3, "Thursday": 4,
    "Friday": 5, "Saturday": 6, "Sunday": 7
}
data['Day of Week'] = data['Day of Week'].map(day_mapping)
```

```
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/3245339923.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Day of Week'] = data['Day of Week'].map(day_mapping)
```

In [12]:
```
data
```

Out[12]:

	Estimated Revenue (USD)	Day of Week	Views	Shares	Video Duration	Dislikes	Likes	Subscribers	Publish Hour	Publish Day
0	0.561	4	23531.0	12.0	201.0	30.0	924.0	51.0	0	2
1	0.648	5	11478.0	5.0	391.0	18.0	322.0	33.0	0	10
2	0.089	2	6153.0	4.0	133.0	20.0	239.0	8.0	0	14
3	0.017	3	4398.0	7.0	14.0	14.0	220.0	2.0	0	29
4	0.000	5	14659.0	7.0	45.0	180.0	602.0	28.0	0	1
...	...	...	...	...	...	...	...	...	...	...
359	8.063	7	10018.0	44.0	779.0	6.0	749.0	16.0	0	25
360	8.705	7	8298.0	26.0	818.0	6.0	587.0	7.0	0	1
361	9.852	1	8487.0	30.0	2233.0	13.0	707.0	14.0	0	16
362	3.858	3	7060.0	21.0	391.0	4.0	483.0	11.0	0	25
363	5.915	5	3890.0	12.0	1875.0	2.0	341.0	-3.0	0	18

364 rows × 12 columns

In [13]:
```
1 data.describe()
```

Out[13]:

	Estimated Revenue (USD)	Day of Week	Views	Shares	Video Duration	Dislikes	Likes
count	364.000000	364.000000	364.000000	364.000000	364.000000	364.000000	364.000000
mean	8.852052	4.071429	128800.101648	252.958791	664.239011	123.961538	5526.733516
std	13.414650	2.050427	118209.844270	363.016405	330.646183	128.311620	4465.210998
min	0.000000	1.000000	2461.000000	1.000000	9.000000	2.000000	121.000000
25%	0.443250	2.000000	27160.500000	38.000000	496.000000	27.000000	1205.500000
50%	4.285000	4.000000	101950.500000	149.000000	613.000000	85.500000	5172.000000
75%	11.476250	6.000000	198169.500000	313.250000	786.500000	179.250000	8504.750000
max	103.117000	7.000000	670990.000000	4190.000000	2311.000000	818.000000	27222.000000

In [14]:
```
1 data['Subscribers'] = data['Subscribers'].apply(lambda x: np.nan if
2 data[data.Subscribers.isna()]
3 #consider to drop / fill na
```

```
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/1709995341.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Subscribers'] = data['Subscribers'].apply(lambda x: np.nan if x < 0 else x)
```

Out[14]:

	Estimated Revenue (USD)	Day of Week	Views	Shares	Video Duration	Dislikes	Likes	Subscribers	Publish Hour	Publish Day	P
363	5.915	5	3890.0	12.0	1875.0	2.0	341.0	NaN	0	18	

In [15]:
```
1 median_subscribers = data['Subscribers'].median()
2 data['Subscribers'].fillna(median_subscribers, inplace=True)
```

```
/var/folders/xc/_0_vbsjd7rq4kvtyyqmmcjxr0000gn/T/ipykernel_976/4237307633.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Subscribers'].fillna(median_subscribers, inplace=True)
```

In [16]:
```
1 data.describe()
```

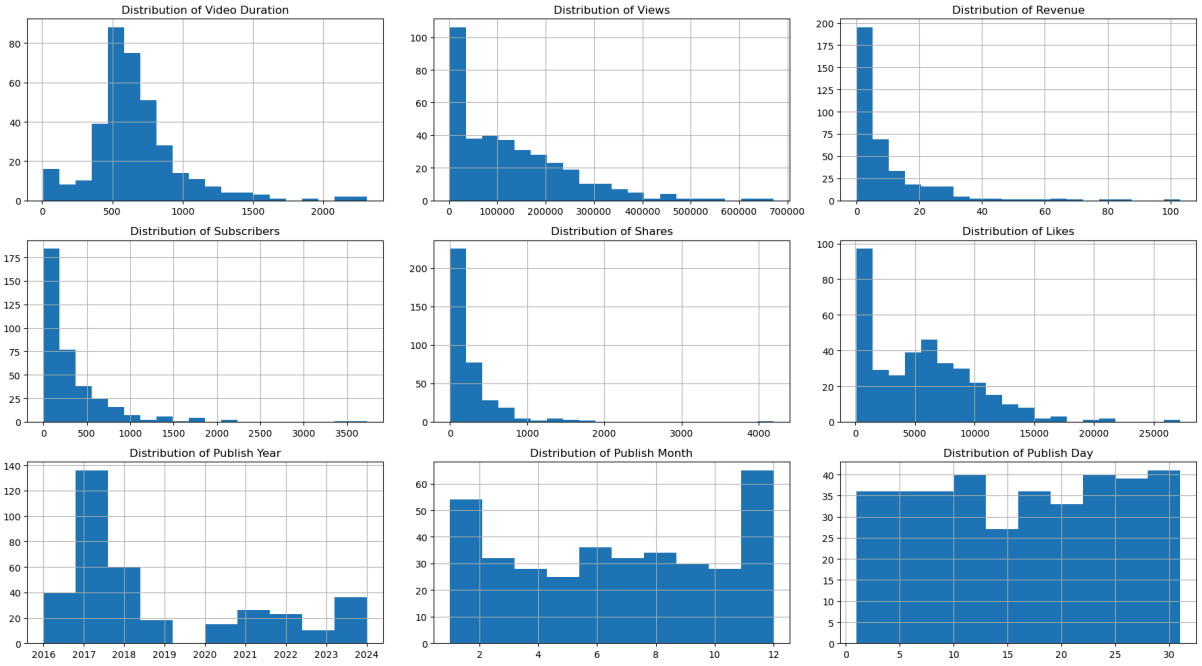
Out[16]:

	Estimated Revenue (USD)	Day of Week	Views	Shares	Video Duration	Dislikes	Likes
count	364.000000	364.000000	364.000000	364.000000	364.000000	364.000000	364.000000
mean	8.852052	4.071429	128800.101648	252.958791	664.239011	123.961538	5526.733516
std	13.414650	2.050427	118209.844270	363.016405	330.646183	128.311620	4465.210998
min	0.000000	1.000000	2461.000000	1.000000	9.000000	2.000000	121.000000
25%	0.443250	2.000000	27160.500000	38.000000	496.000000	27.000000	1205.500000
50%	4.285000	4.000000	101950.500000	149.000000	613.000000	85.500000	5172.000000
75%	11.476250	6.000000	198169.500000	313.250000	786.500000	179.250000	8504.750000
max	103.117000	7.000000	670990.000000	4190.000000	2311.000000	818.000000	27222.000000

In [17]:
```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
```

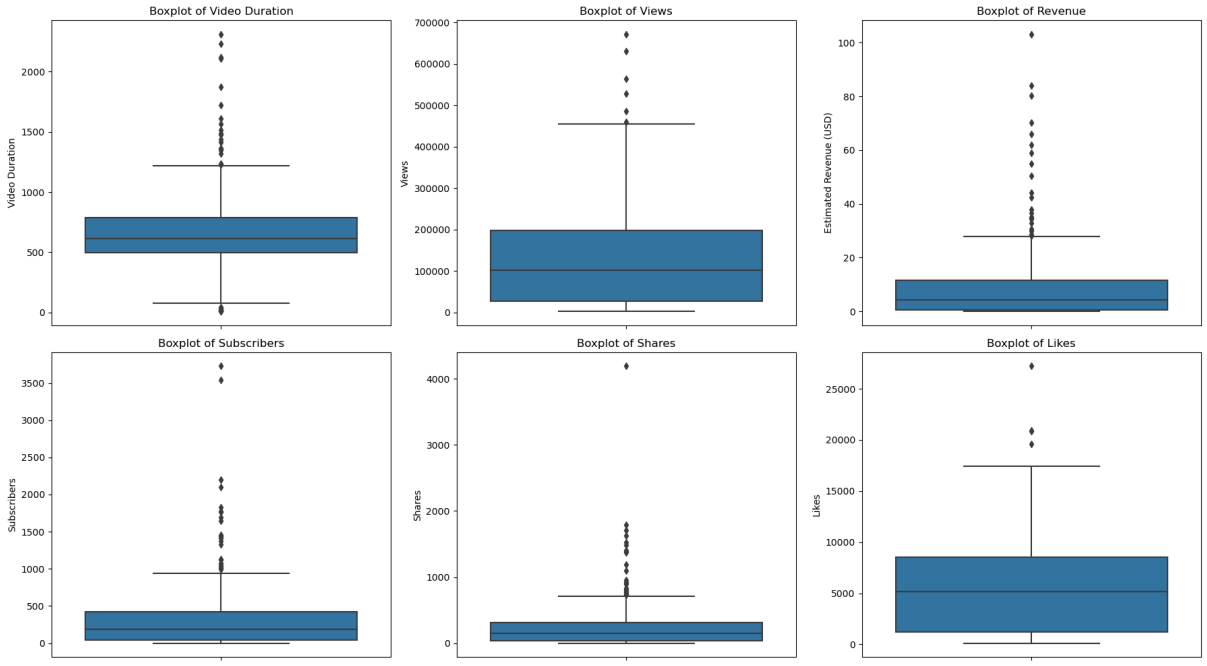
In [18]:

```
fig, axes = plt.subplots(3,3, figsize=(18, 10))
data['Video Duration'].hist(ax=axes[0, 0], bins=20)
axes[0, 0].set_title('Distribution of Video Duration')
data['Views'].hist(ax=axes[0, 1], bins=20)
axes[0, 1].set_title('Distribution of Views')
data['Estimated Revenue (USD)'].hist(ax=axes[0, 2], bins=20)
axes[0, 2].set_title('Distribution of Revenue')
data['Subscribers'].hist(ax=axes[1, 0], bins=20)
axes[1, 0].set_title('Distribution of Subscribers')
data['Shares'].hist(ax=axes[1, 1], bins=20)
axes[1, 1].set_title('Distribution of Shares')
data['Likes'].hist(ax=axes[1, 2], bins=20)
axes[1, 2].set_title('Distribution of Likes')
data['Publish Year'].hist(ax=axes[2, 0])
axes[2, 0].set_title('Distribution of Publish Year')
data['Publish Month'].hist(ax=axes[2, 1])
axes[2, 1].set_title('Distribution of Publish Month')
data['Publish Day'].hist(ax=axes[2, 2])
axes[2, 2].set_title('Distribution of Publish Day')
plt.tight_layout()
plt.show()
#skewed distribution, so should be StandardScaler
```



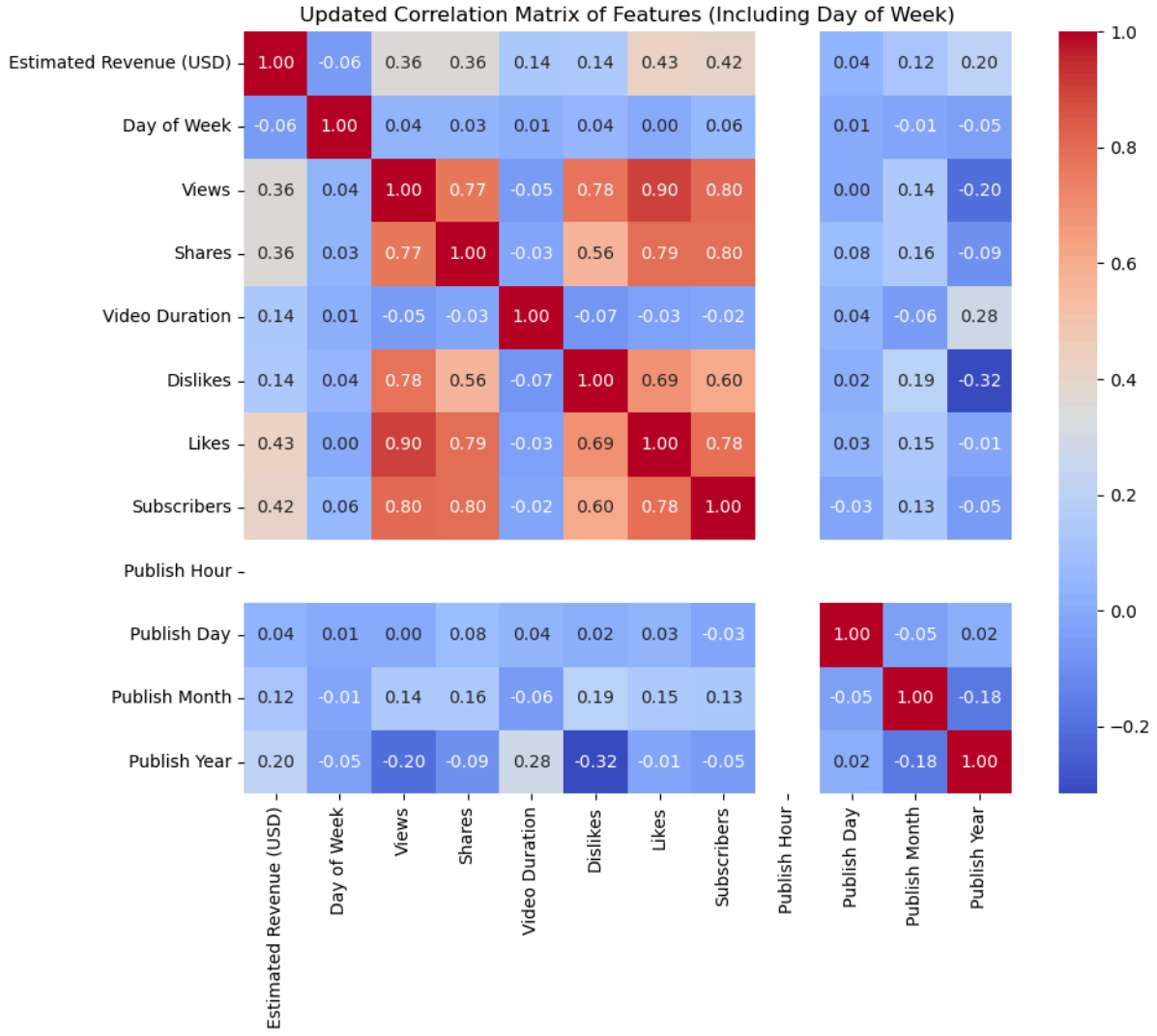
In [19]:

```
# Creating boxplots for the same features to visualize outliers
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
sns.boxplot(data=data, y='Video Duration', ax=axes[0, 0])
axes[0, 0].set_title('Boxplot of Video Duration')
sns.boxplot(data=data, y='Views', ax=axes[0, 1])
axes[0, 1].set_title('Boxplot of Views')
sns.boxplot(data=data, y='Estimated Revenue (USD)', ax=axes[0, 2])
axes[0, 2].set_title('Boxplot of Revenue')
sns.boxplot(data=data, y='Subscribers', ax=axes[1, 0])
axes[1, 0].set_title('Boxplot of Subscribers')
sns.boxplot(data=data, y='Shares', ax=axes[1, 1])
axes[1, 1].set_title('Boxplot of Shares')
sns.boxplot(data=data, y='Likes', ax=axes[1, 2])
axes[1, 2].set_title('Boxplot of Likes')
plt.tight_layout()
plt.show()
```



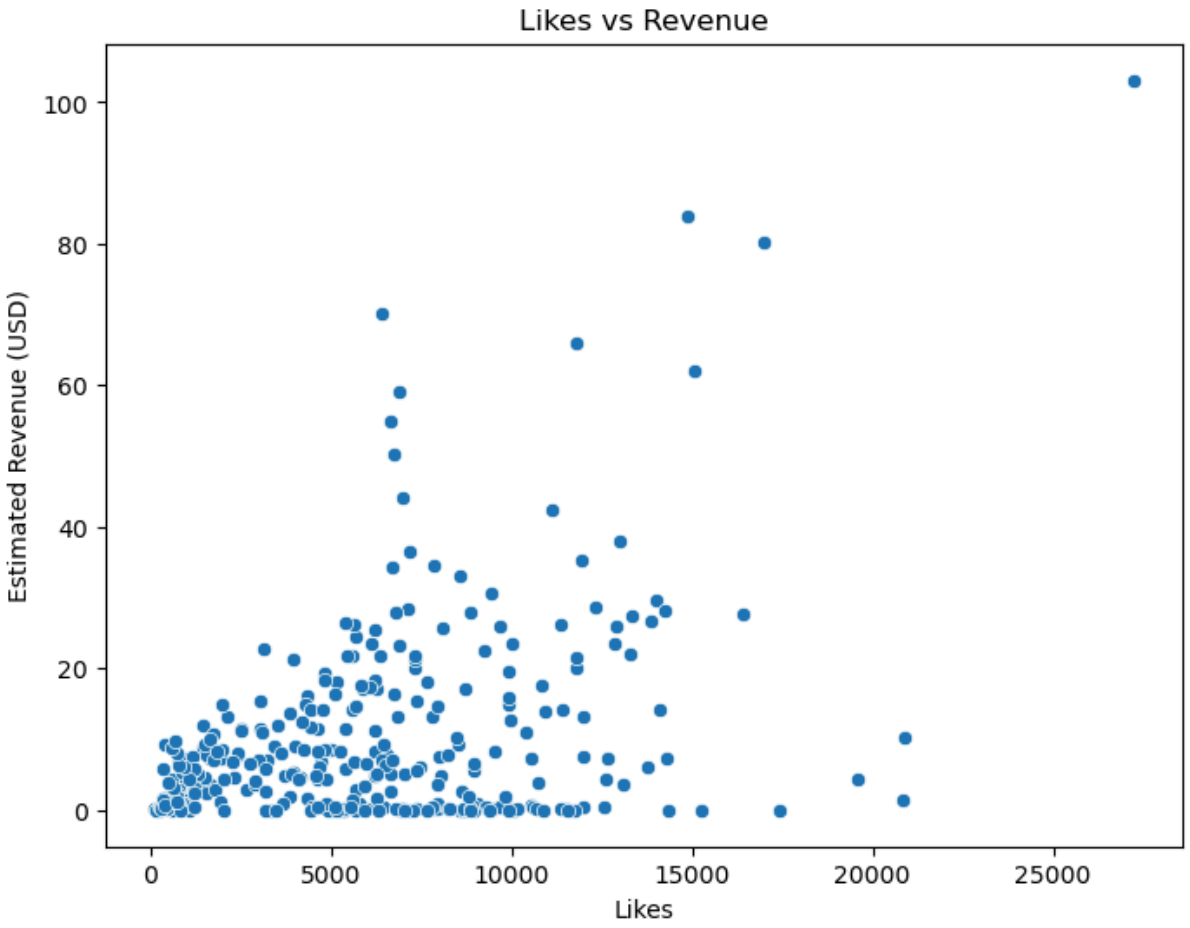
In [20]:

```
corr_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", cba
plt.title("Updated Correlation Matrix of Features (Including Day of \
plt.show()
```



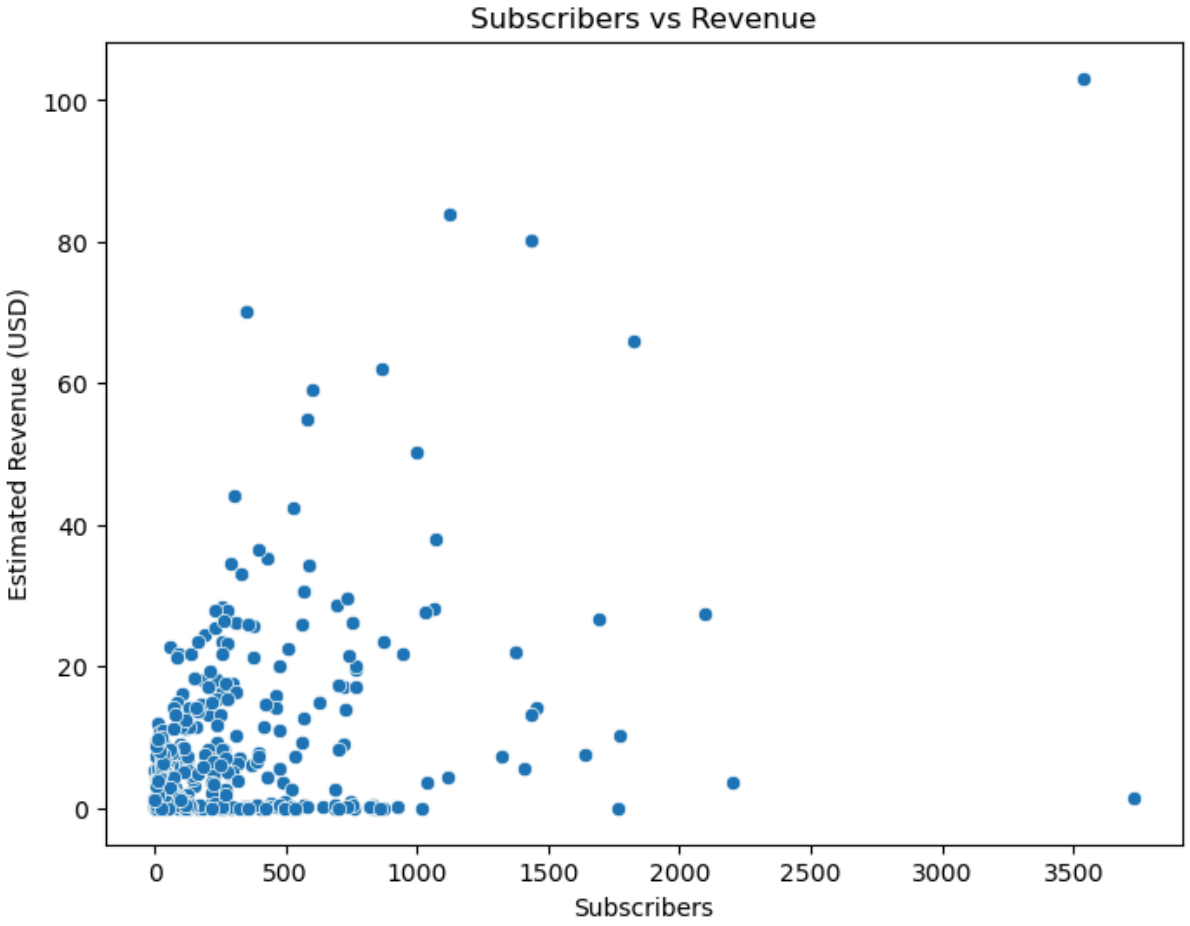
In [21]:

```
# Scatterplot: Views vs Revenue
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Likes', y='Estimated Revenue (USD)', data=data)
plt.title("Likes vs Revenue")
plt.xlabel("Likes")
plt.ylabel("Estimated Revenue (USD)")
plt.show()
```



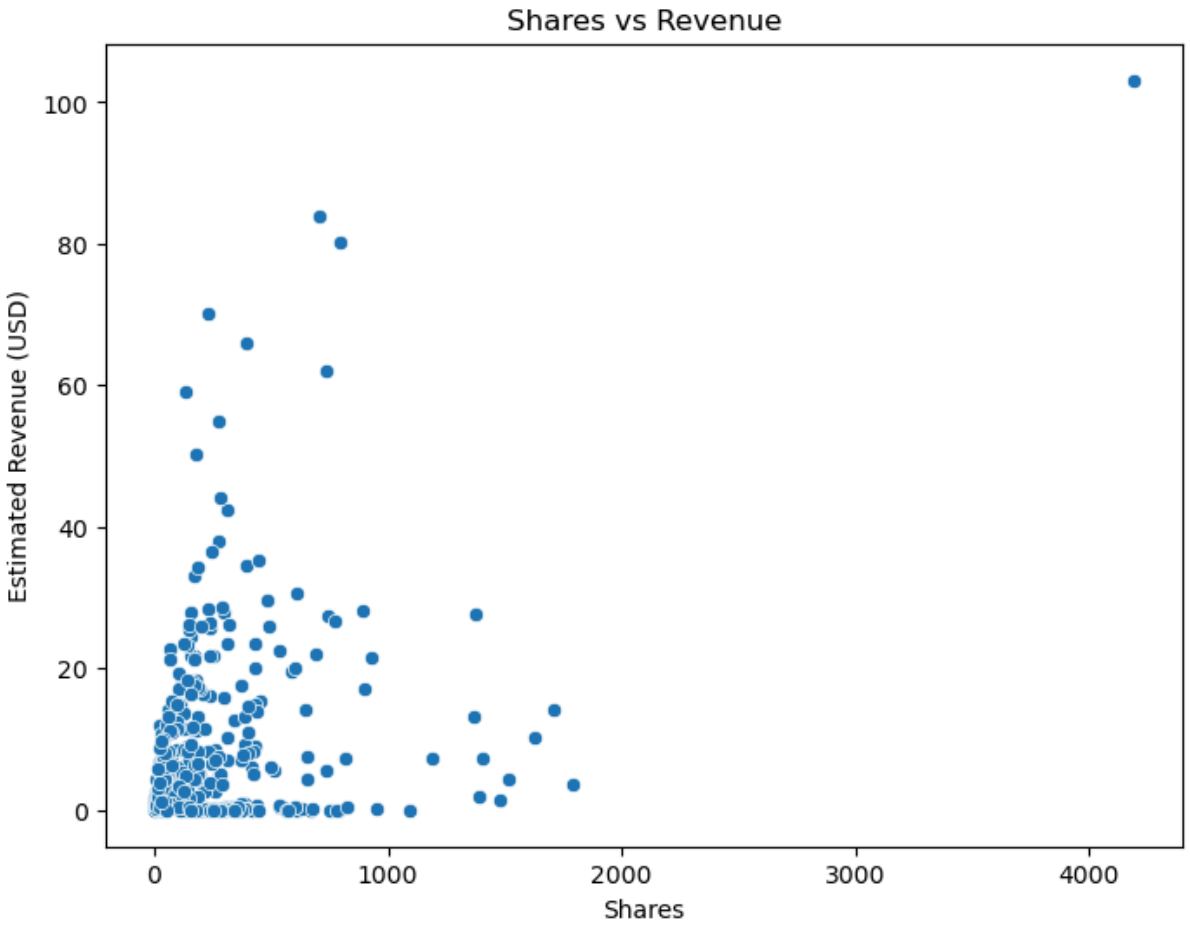
In [22]:

```
# Scatterplot: Views vs Revenue
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Subscribers', y='Estimated Revenue (USD)', data=d
plt.title("Subscribers vs Revenue")
plt.xlabel("Subscribers")
plt.ylabel("Estimated Revenue (USD)")
plt.show()
```

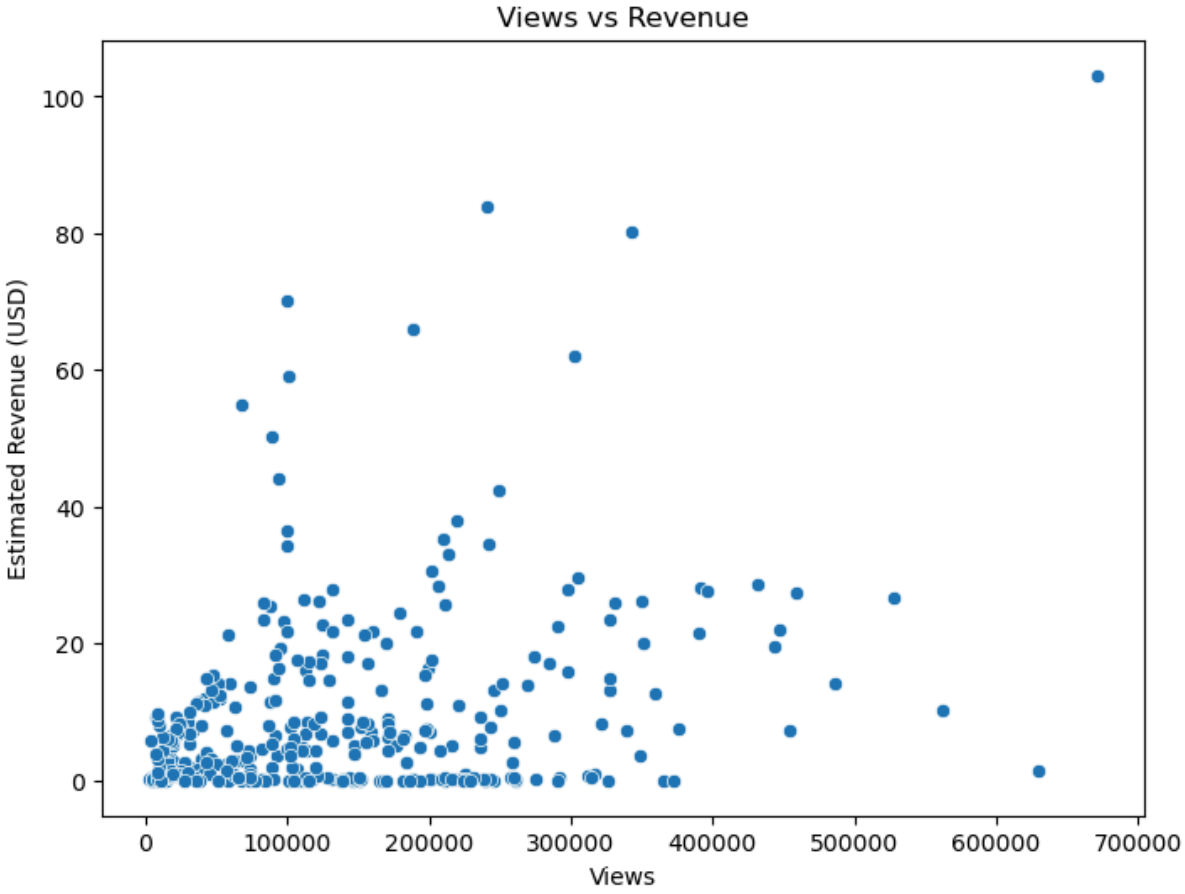


In [23]:

```
# Scatterplot: Views vs Revenue
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Shares', y='Estimated Revenue (USD)', data=data)
plt.title("Shares vs Revenue")
plt.xlabel("Shares")
plt.ylabel("Estimated Revenue (USD)")
plt.show()
```



In [24]:
```
# Scatterplot: Views vs Revenue
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Views', y='Estimated Revenue (USD)', data=data)
plt.title("Views vs Revenue")
plt.xlabel("Views")
plt.ylabel("Estimated Revenue (USD)")
plt.show()
# A clear positive relationship is observed, confirming that higher
# Outliers with very high revenue and views are present.
```



Model with Outliers

In [25]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, f1_score
```

In [26]:
```
data = pd.get_dummies(data, columns=['Day of Week'],prefix = 'Day')
```

In [27]:
```
1 data
```

Out[27]:

	Estimated Revenue (USD)	Views	Shares	Video Duration	Dislikes	Likes	Subscribers	Publish Hour	Publish Day	Publish Month
0	0.561	23531.0	12.0	201.0	30.0	924.0	51.0	0	2	6
1	0.648	11478.0	5.0	391.0	18.0	322.0	33.0	0	10	6
2	0.089	6153.0	4.0	133.0	20.0	239.0	8.0	0	14	6
3	0.017	4398.0	7.0	14.0	14.0	220.0	2.0	0	29	6
4	0.000	14659.0	7.0	45.0	180.0	602.0	28.0	0	1	7
...	...	...	...	...	...	...	...	...	...	...
359	8.063	10018.0	44.0	779.0	6.0	749.0	16.0	0	25	8
360	8.705	8298.0	26.0	818.0	6.0	587.0	7.0	0	1	9
361	9.852	8487.0	30.0	2233.0	13.0	707.0	14.0	0	16	9
362	3.858	7060.0	21.0	391.0	4.0	483.0	11.0	0	25	9
363	5.915	3890.0	12.0	1875.0	2.0	341.0	185.0	0	18	10

364 rows × 18 columns

In [28]:
```
1  features= [
2      "Video Duration", "Views", "Subscribers", "Shares", "Likes", "Di
3      "Publish Hour", "Publish Day", "Publish Month", "Publish Year"
4  ]
5  target = "Estimated Revenue (USD)"
6
7  X = data[features]
8  y = data[target]
9
10 year_counts = data['Publish Year'].value_counts()
11 year_weights = 1 / year_counts
12 data['Sample Weight'] = data['Publish Year'].map(year_weights)
13
14 X_train, X_test, y_train, y_test, weights_train, weights_test = trai
15
16 scaler = StandardScaler()
17 X_train_scaled = scaler.fit_transform(X_train)
18 X_test_scaled = scaler.transform(X_test)
```

In [29]:

```python
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoosting
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(random_state=42),
    "Gradient Boosting Regressor": GradientBoostingRegressor(random_
    "Support Vector Regressor (SVR)": SVR()
}

param_grids = {
    "Linear Regression": {
        "fit_intercept": [True, False]
    },
    "Random Forest": {
        "n_estimators": [50, 100, 200],
        "max_depth": [None, 10, 20],
        "min_samples_split": [2, 5]
    },
    "Gradient Boosting Regressor": {
        "n_estimators": [50, 100, 200],
        "learning_rate": [0.01, 0.1, 0.2],
        "max_depth": [3, 5, 7]
    },
    "Support Vector Regressor (SVR)": {
        "C": [0.1, 1, 10],
        "gamma": [0.01, 0.1, 1],
        "kernel": ["rbf", "linear"]
    }
}

results = {}

for model_name, model in models.items():
    print(f"Running GridSearchCV for {model_name}...")

    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grids[model_name],
        cv=5,
        scoring='neg_mean_squared_error',
        n_jobs=-1,
        verbose=2
    )

    grid_search.fit(X_train_scaled, y_train, sample_weight=weights_t

    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test_scaled)
```

```
58
59      mse = mean_squared_error(y_test, y_pred)
60      r2 = r2_score(y_test, y_pred)
61
62      results[model_name] = {
63          "Best Params": grid_search.best_params_,
64          "MSE": mse,
65          "R2": r2
66      }
67
68  results_df = pd.DataFrame(results).T
69  results_df
```

```
Running GridSearchCV for Linear Regression...
Fitting 5 folds for each of 2 candidates, totalling 10 fits
Running GridSearchCV for Random Forest...
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Running GridSearchCV for Gradient Boosting Regressor...
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Running GridSearchCV for Support Vector Regressor (SVR)...
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

Out[29]:

	Best Params	MSE	R2
Linear Regression	{'fit_intercept': True}	114.698239	-0.317367
Random Forest	{'max_depth': None, 'min_samples_split': 5, 'n...	97.535223	-0.120241
Gradient Boosting Regressor	{'learning_rate': 0.01, 'max_depth': 5, 'n_est...	127.272812	-0.461792
Support Vector Regressor (SVR)	{'C': 10, 'gamma': 0.01, 'kernel': 'linear'}	73.211304	0.159131

Model without outliers

In [30]:

```python
def remove_outliers(df, columns):
    for col in columns:
        if df[col].dtype in ['float64', 'int64']:
            q1 = df[col].quantile(0.25)
            q3 = df[col].quantile(0.75)
            iqr = q3 - q1
            lower_bound = q1 - 1.5 * iqr
            upper_bound = q3 + 1.5 * iqr
            df = df[(df[col] >= lower_bound) & (df[col] <= upper_bou
    return df

columns_to_check = features + [target]
data_no_outliers_full = remove_outliers(data, columns_to_check)
data_no_outliers_full.shape

year_counts_no = data_no_outliers_full['Publish Year'].value_counts(
year_weights_no = 1 / year_counts_no
data_no_outliers_full['Sample Weight'] = data_no_outliers_full['Publ

X_no_outliers_full = data_no_outliers_full[features]
y_no_outliers_full = data_no_outliers_full[target]

X_train_no_outliers_full, X_test_no_outliers_full, y_train_no_outlie
    X_no_outliers_full, y_no_outliers_full,data_no_outliers_full['Sa
)

X_train_no_outliers_full_scaled = scaler.fit_transform(X_train_no_ou
X_test_no_outliers_full_scaled = scaler.transform(X_test_no_outliers_

models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(random_state=42),
    "Gradient Boosting Regressor": GradientBoostingRegressor(random_
    "Support Vector Regressor (SVR)": SVR()
}

param_grids = {
    "Linear Regression": {
        "fit_intercept": [True, False]
    },
    "Random Forest": {
    "n_estimators": [100, 200],
    "max_depth": [10, 20]
    },
    "Gradient Boosting Regressor": {
        "n_estimators": [50, 100, 200],
        "learning_rate": [0.01, 0.1, 0.2],
        "max_depth": [3, 5, 7]
    },
    "Support Vector Regressor (SVR)": {
        "C": [0.1, 1, 10],
        "gamma": [0.01, 0.1, 1],
        "kernel": ["rbf"]
    }
}

# Results dictionary
```

```python
grid_results = {}

for model_name, model in models.items():
    print(f"Running GridSearchCV for {model_name}...")

    # Define GridSearchCV
    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grids.get(model_name, {}),
        cv=5,
        scoring="neg_mean_squared_error",
        n_jobs=-1,
        verbose=2
    )

    # Fit the model with sample weights
    grid_search.fit(
        X_train_no_outliers_full_scaled,
        y_train_no_outliers_full,
        sample_weight=weights_train_no_outliers_full
    )

    # Predict using the best estimator
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test_no_outliers_full_scaled)

    # Evaluate model performance
    mse = mean_squared_error(y_test_no_outliers_full, y_pred)
    r2 = r2_score(y_test_no_outliers_full, y_pred)

    # Store results
    grid_results[model_name] = {
        "Best Params": grid_search.best_params_,
        "MSE": mse,
        "R2": r2
    }

# Display results
results_df_no = pd.DataFrame(grid_results).T
results_df_no
```

```
Running GridSearchCV for Linear Regression...
Fitting 5 folds for each of 2 candidates, totalling 10 fits
Running GridSearchCV for Random Forest...
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Running GridSearchCV for Gradient Boosting Regressor...
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Running GridSearchCV for Support Vector Regressor (SVR)...
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

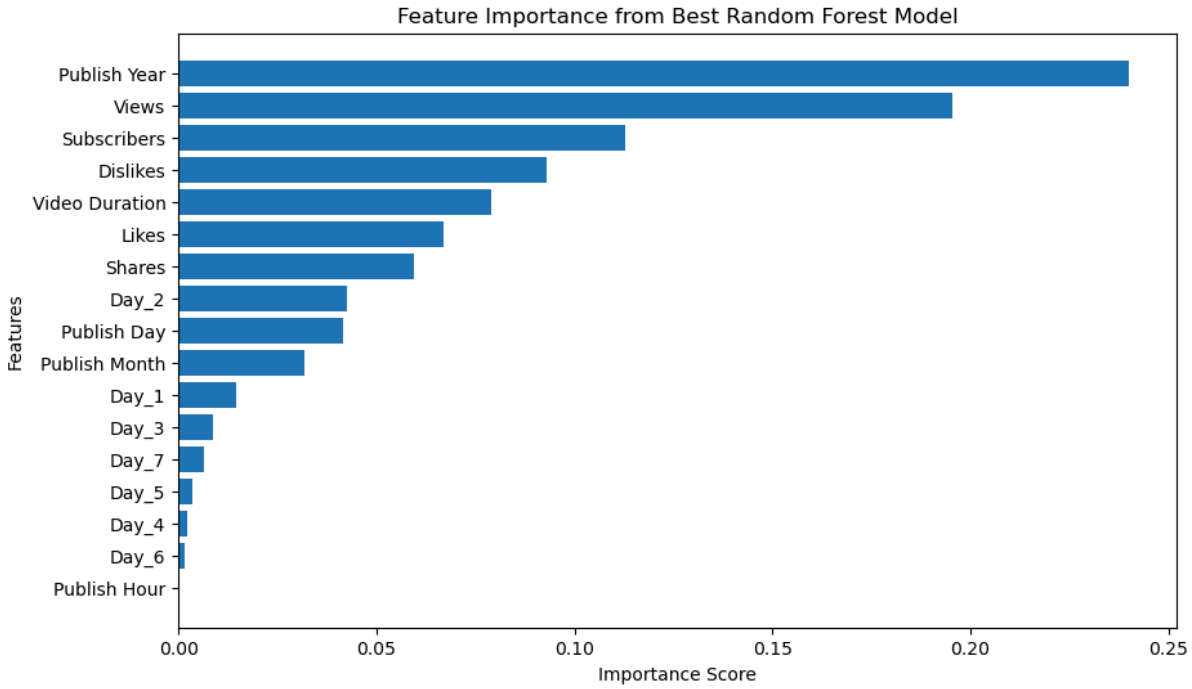
Out[30]:

	Best Params	MSE	R2
Linear Regression	{'fit_intercept': True}	29.80553	0.176494
Random Forest	{'max_depth': 20, 'n_estimators': 200}	16.970346	0.531121
Gradient Boosting Regressor	{'learning_rate': 0.1, 'max_depth': 3, 'n_esti...	18.99129	0.475284
Support Vector Regressor (SVR)	{'C': 1, 'gamma': 1, 'kernel': 'rbf'}	39.400743	-0.088616

In [31]:

```
best_rf_model = grid_results["Random Forest"]["Best Params"]

best_rf = models["Random Forest"].set_params(**best_rf_model)

best_rf.fit(
    X_train_no_outliers_full_scaled,
    y_train_no_outliers_full,
    sample_weight=weights_train_no_outliers_full
)

feature_importances = best_rf.feature_importances_

importance_df = pd.DataFrame({
    "Feature": features,
    "Importance": feature_importances
}).sort_values(by="Importance", ascending=False)

print(importance_df)

plt.figure(figsize=(10, 6))
plt.barh(importance_df["Feature"], importance_df["Importance"], alig
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance from Best Random Forest Model")
plt.gca().invert_yaxis()
plt.show()
```

```
          Feature  Importance
16    Publish Year    0.240142
1            Views    0.195459
2      Subscribers    0.112868
5         Dislikes    0.092844
0   Video Duration    0.078948
4            Likes    0.067113
3           Shares    0.059588
7            Day_2    0.042483
14     Publish Day    0.041652
15   Publish Month    0.031682
6            Day_1    0.014617
8            Day_3    0.008842
12           Day_7    0.006451
10           Day_5    0.003492
9            Day_4    0.002147
11           Day_6    0.001673
13    Publish Hour    0.000000
```


Feature Importance from Best Random Forest Model
Publish Year
Views
Subscribers
Dislikes
Video Duration
Likes
Shares
Day_2
Publish Day
Publish Month
Day_1
Day_3
Day_7
Day_5
Day_4
Day_6
Publish Hour
Features
0.00
0.05
0.10
0.15
0.20
0.25
Importance Score

Logistic Regression with outliers

In [32]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier

X_log = data.drop(columns=['Estimated Revenue (USD)','Sample Weight'
y_log = (data['Estimated Revenue (USD)'] > data['Estimated Revenue (

year_counts = data['Publish Year'].value_counts()
year_weights = 1 / year_counts
data['Sample Weight'] = data['Publish Year'].map(year_weights)

X_train_log, X_test_log, y_train_log, y_test_log,weights_train_log, 

scaler = StandardScaler()
X_train_scaled_log = scaler.fit_transform(X_train_log)
X_test_scaled_log = scaler.transform(X_test_log)



param_grids = {
    "Logistic Regression": {
        "C": [0.1, 1, 10],
        "penalty": ["l2"],
        "solver": ["lbfgs"]
    },
    "Random Forest": {
        "n_estimators": [50, 100, 200],
        "max_depth": [None, 10, 20],
        "min_samples_split": [2, 5],
        "min_samples_leaf": [1, 2]
    },
    "Support Vector Classifier (SVC)": {
        "C": [0.1, 1, 10],
        "gamma": [0.01, 0.1, 1],
        "kernel": ["rbf"]
    },
    "Gradient Boosting Classifier": {
        "n_estimators": [50, 100, 200],
        "learning_rate": [0.01, 0.1, 0.2],
        "max_depth": [3, 5, 7]
    }
}

# Models
models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Classifier (SVC)": SVC(probability=True, random_
    "Gradient Boosting Classifier": GradientBoostingClassifier(rando
}

# Results dictionary
grid_results_binary = {}

# Perform GridSearchCV for each model
```

```python
for model_name, model in models.items():
    print(f"Running GridSearchCV for {model_name}...")

    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grids[model_name],
        cv=5,
        scoring="accuracy",
        n_jobs=-1,
        verbose=2
    )

    # Fit the model with sample weights
    grid_search.fit(X_train_scaled_log, y_train_log, sample_weight=w

    # Predict using the best estimator
    best_model = grid_search.best_estimator_
    y_pred_log = (best_model.predict_proba(X_test_scaled_log)[:, 1] :

    # Evaluate metrics
    accuracy = accuracy_score(y_test_log, y_pred_log)
    precision = precision_score(y_test_log, y_pred_log)
    recall = recall_score(y_test_log, y_pred_log)
    f1 = f1_score(y_test_log, y_pred_log)

    # Store results
    grid_results_binary[model_name] = {
        "Best Params": grid_search.best_params_,
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1
    }

results_df_log = pd.DataFrame(grid_results_binary).T
results_df_log
```

```
Running GridSearchCV for Logistic Regression...
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Running GridSearchCV for Random Forest...
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Running GridSearchCV for Support Vector Classifier (SVC)...
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Running GridSearchCV for Gradient Boosting Classifier...
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

Out[32]:

	Best Params	Accuracy	Precision	Recall	F1 Score
Logistic Regression	{'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}	0.684932	0.6	0.909091	0.722892
Random Forest	{'max_depth': None, 'min_samples_leaf': 2, 'mi...	0.808219	0.756757	0.848485	0.8
Support Vector Classifier (SVC)	{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}	0.684932	0.638889	0.69697	0.666667
Gradient Boosting Classifier	{'learning_rate': 0.1, 'max_depth': 5, 'n_esti...	0.767123	0.735294	0.757576	0.746269

Logistic Regression without outliers

In [33]:

```python
# Ensure the necessary features and target are defined
features = ["Video Duration", "Views", "Subscribers", "Shares", "Lik
    "Publish Hour", "Publish Day", "Publish Month", "Publish Year"]

# Convert target into binary classification
data['binary_target'] = (data['Estimated Revenue (USD)'] > data['Est

columns_to_check = features + ['binary_target']
data_no_outliers_full = remove_outliers(data, columns_to_check)

year_counts_without = data_no_outliers_full['Publish Year'].value_co
year_weights_without = 1 / year_counts_without
data_no_outliers_full['Sample Weight'] = data_no_outliers_full['Publ

X_no_outliers_full = data_no_outliers_full[features]
y_no_outliers_full = data_no_outliers_full['binary_target']

# Split data
X_train_no_outliers_full, X_test_no_outliers_full, y_train_no_outlie
    X_no_outliers_full, y_no_outliers_full,data_no_outliers_full['Sa
)

# Scaling
X_train_no_outliers_full_scaled = scaler.fit_transform(X_train_no_ou
X_test_no_outliers_full_scaled = scaler.transform(X_test_no_outliers

models = {
    "Logistic Regression": LogisticRegression(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Classifier (SVC)": SVC(probability=True, random_
    "Gradient Boosting Classifier": GradientBoostingClassifier(randc
}

# Define models
param_grids = {
    "Logistic Regression": {
        "C": [0.1, 1, 10],
        "penalty": ["l2"],
        "solver": ["lbfgs"]
    },
    "Random Forest": {
        "n_estimators": [50, 100, 200],
        "max_depth": [None, 10, 20],
        "min_samples_split": [2, 5],
        "min_samples_leaf": [1, 2]
    },
    "Support Vector Classifier (SVC)": {
        "C": [0.1, 1, 10],
        "gamma": [0.01, 0.1, 1],
        "kernel": ["rbf"]
    },
    "Gradient Boosting Classifier": {
        "n_estimators": [50, 100, 200],
        "learning_rate": [0.01, 0.1, 0.2],
        "max_depth": [3, 5, 7]
    }
}
```

```python

# Results dictionary
grid_results = {}

# Perform GridSearchCV for each model
for model_name, model in models.items():
    print(f"Running GridSearchCV for {model_name}...")

    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grids[model_name],
        cv=5,
        scoring="accuracy",
        n_jobs=-1,
        verbose=2
    )

    # Fit the model with sample weights
    grid_search.fit(
        X_train_no_outliers_full_scaled,
        y_train_no_outliers_full,
        sample_weight=weights_train_no_outliers_full
    )

    # Predict using the best estimator
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test_no_outliers_full_scaled)

    # Evaluate metrics
    accuracy = accuracy_score(y_test_no_outliers_full, y_pred)
    precision = precision_score(y_test_no_outliers_full, y_pred)
    recall = recall_score(y_test_no_outliers_full, y_pred)
    f1 = f1_score(y_test_no_outliers_full, y_pred)

    # Store results
    grid_results[model_name] = {
        "Best Params": grid_search.best_params_,
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F-1 Score": f1
    }

# Display results
import pandas as pd
results_df_log_no = pd.DataFrame(grid_results).T
results_df_log_no
```

```
Running GridSearchCV for Logistic Regression...
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Running GridSearchCV for Random Forest...
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Running GridSearchCV for Support Vector Classifier (SVC)...
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Running GridSearchCV for Gradient Boosting Classifier...
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

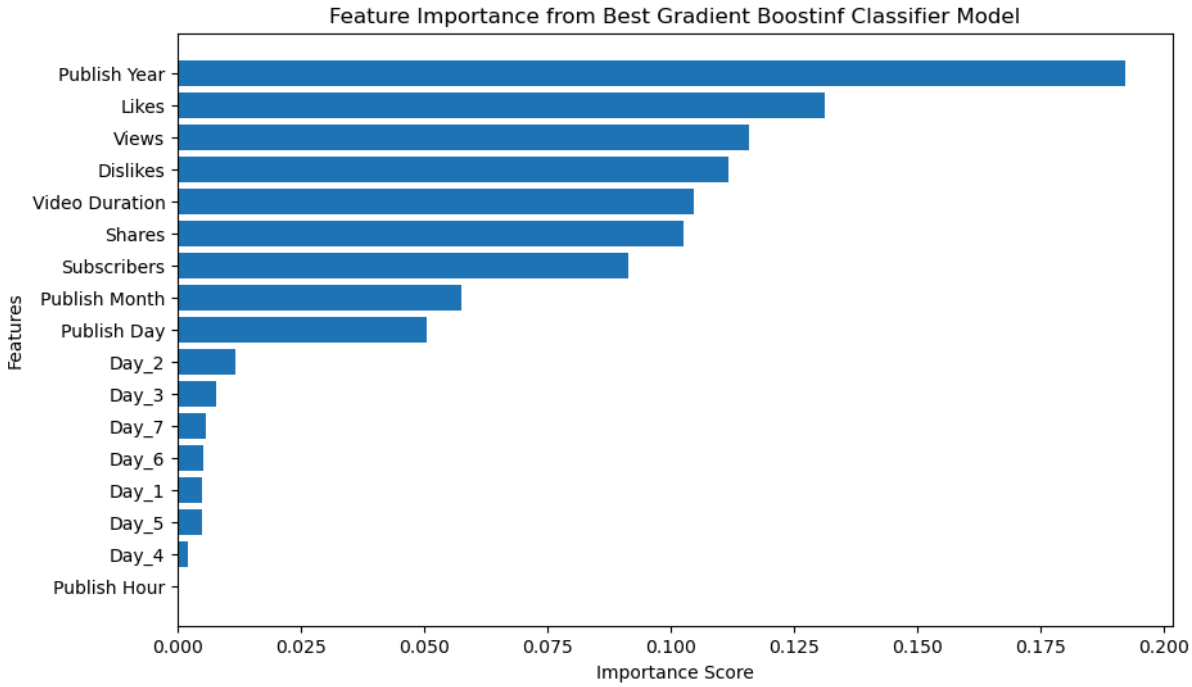
Out[33]:

	Best Params	Accuracy	Precision	Recall	F-1 Score
Logistic Regression	{'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'}	0.576271	0.5	0.48	0.489796
Random Forest	{'max_depth': None, 'min_samples_leaf': 2, 'mi...	0.813559	0.71875	0.92	0.807018
Support Vector Classifier (SVC)	{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}	0.644068	0.576923	0.6	0.588235
Gradient Boosting Classifier	{'learning_rate': 0.2, 'max_depth': 5, 'n_esti...	0.830508	0.758621	0.88	0.814815

In [34]:

```
best_rf_model = grid_results["Random Forest"]["Best Params"]

best_rf = models["Random Forest"].set_params(**best_rf_model)

best_rf.fit(
    X_train_no_outliers_full_scaled,
    y_train_no_outliers_full,
    sample_weight=weights_train_no_outliers_full
)

feature_importances = best_rf.feature_importances_

importance_df = pd.DataFrame({
    "Feature": features,
    "Importance": feature_importances
}).sort_values(by="Importance", ascending=False)

print(importance_df)

plt.figure(figsize=(10, 6))
plt.barh(importance_df["Feature"], importance_df["Importance"], alig
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance from Best Gradient Boostinf Classifier
plt.gca().invert_yaxis()
plt.show()
```

```
          Feature  Importance
16    Publish Year    0.192297
4            Likes    0.131159
1            Views    0.115948
5         Dislikes    0.111621
0   Video Duration    0.104707
3           Shares    0.102617
2      Subscribers    0.091532
15   Publish Month    0.057489
14     Publish Day    0.050586
7            Day_2    0.011624
8            Day_3    0.007761
12           Day_7    0.005727
11           Day_6    0.005048
6            Day_1    0.005018
10           Day_5    0.004840
9            Day_4    0.002025
13    Publish Hour    0.000000
```

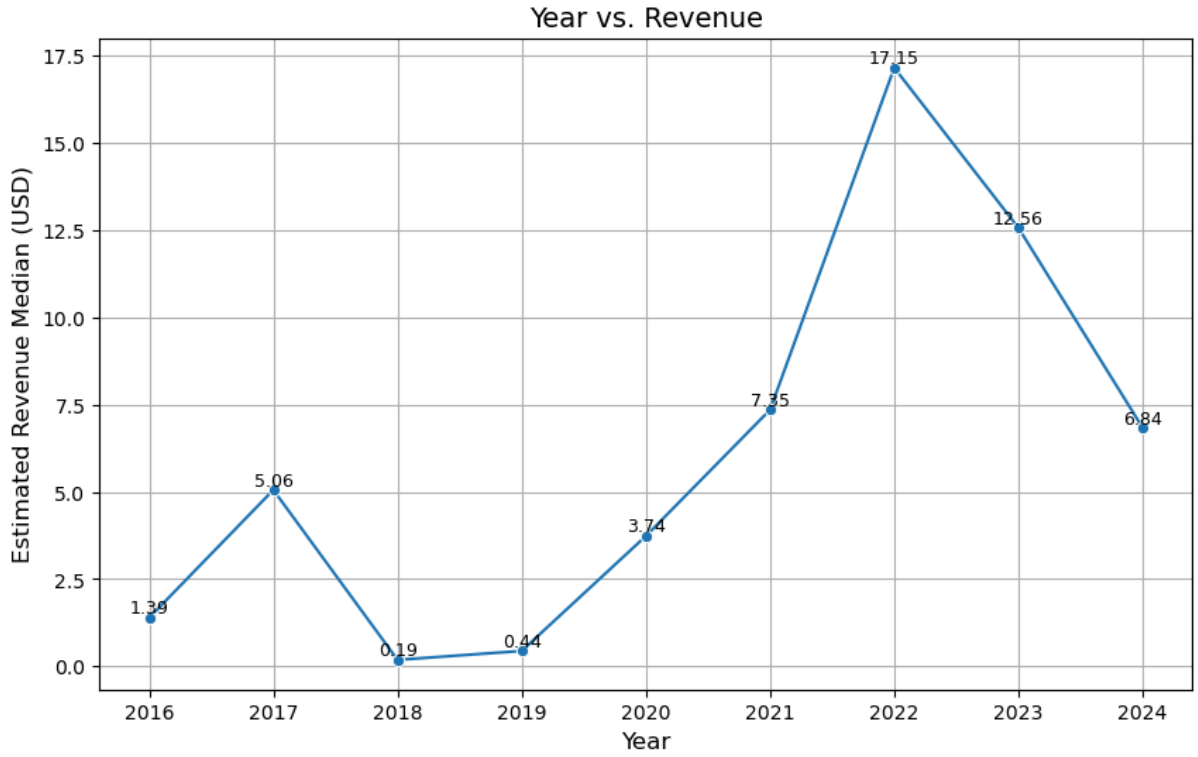

Feature Importance from Best Gradient Boostinf Classifier Model
Publish Year
Likes
Views
Dislikes
Video Duration
Shares
Subscribers
Publish Month
Publish Day
Day_2
Day_3
Day_7
Day_6
Day_1
Day_5
Day_4
Publish Hour
Features
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
Importance Score

Recommendation

In [35]:

```
df_year = data_no_outliers_full.groupby('Publish Year')["Estimated R

plt.figure(figsize=(10, 6))
sns.lineplot(x=df_year["Publish Year"], y=df_year["Estimated Revenue

for i, row in df_year.iterrows():
    plt.text(row["Publish Year"], row["Estimated Revenue (USD)"],
             f'{row["Estimated Revenue (USD)"]:.2f}',
             ha='center', va='bottom', fontsize=9)

plt.title("Year vs. Revenue", fontsize=14)
plt.xlabel("Year", fontsize=12)
plt.ylabel("Estimated Revenue Median (USD)", fontsize=12)
plt.grid(True)
plt.show()
```

```
/Users/aaron/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.p
y:1119: FutureWarning: use_inf_as_na option is deprecated and will be r
emoved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/Users/aaron/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.p
y:1119: FutureWarning: use_inf_as_na option is deprecated and will be r
emoved in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



External Factor: Policy Factor? Time Factor? Internal Factor: Improvement of recommendation system? Category of video? Specific population who watched the video

Assumption: elder people who also watch youtube but previously they did not. Some youtuber sign up and post video and their video attract more revenue.