DIMOND PRICE PREDICTION

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler
import xgboost as xgb
plt.style.use('fivethirtyeight')
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
diamond_df = pd.read_csv('diamonds.csv')
diamond_df.head()
```

Out[2]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

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

Out[3]:
```
(53940, 10)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object 
 2   color    53940 non-null  object 
 3   clarity  53940 non-null  object 
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64  
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

In [5]:
```
diamond_df.isnull().sum()
```

Out[5]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

In [6]:
```
diamond_df.columns
```

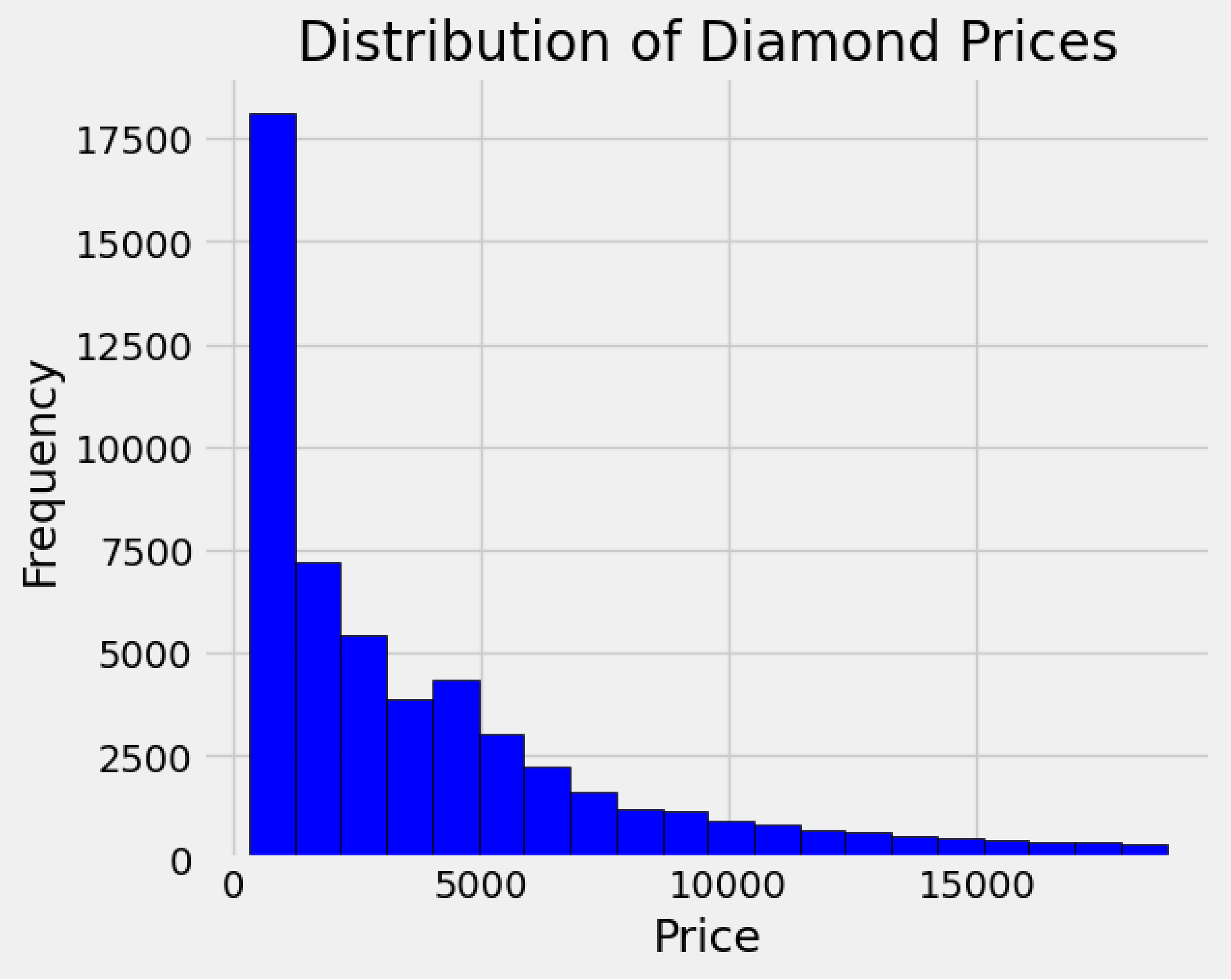
Out[6]:
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
       'z'],
      dtype='object')
```

In [8]:
```
diamond_df.describe()
```

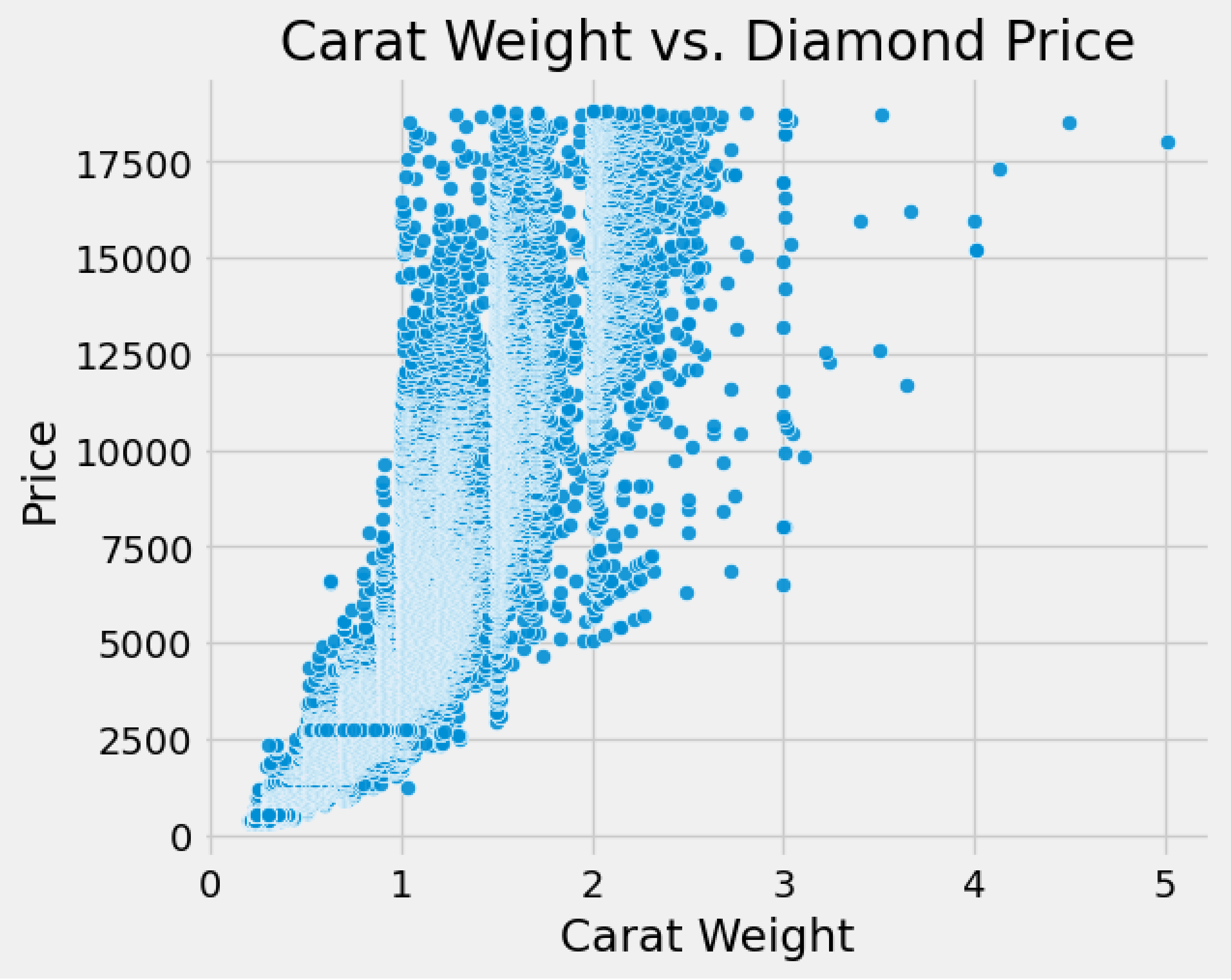
Out[8]:

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

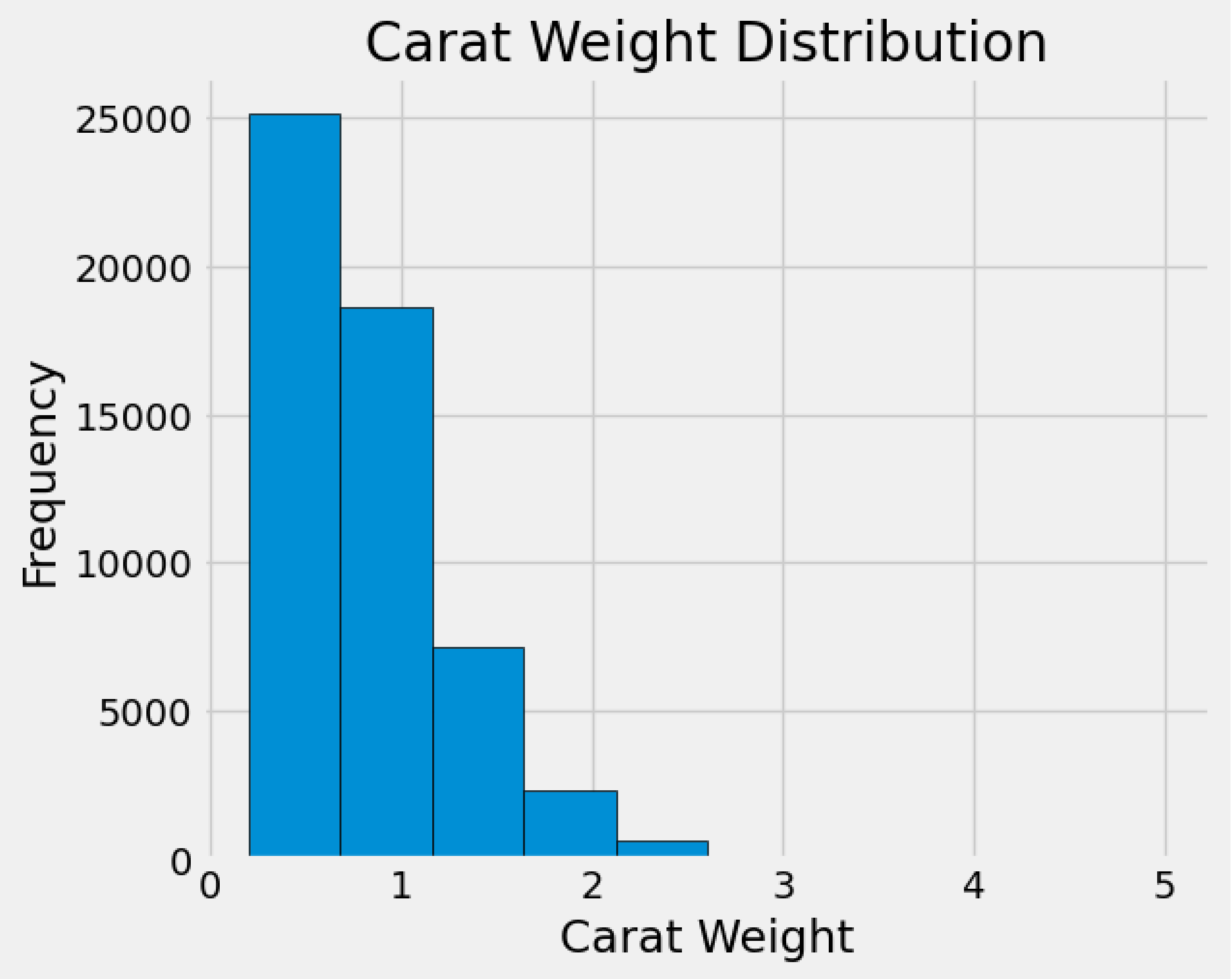
In [16]:
```
plt.figure(figsize=(6,5))
plt.hist(diamond_df['price'], bins=20, color='b', edgecolor='k');
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Distribution of Diamond Prices')
plt.show()
```



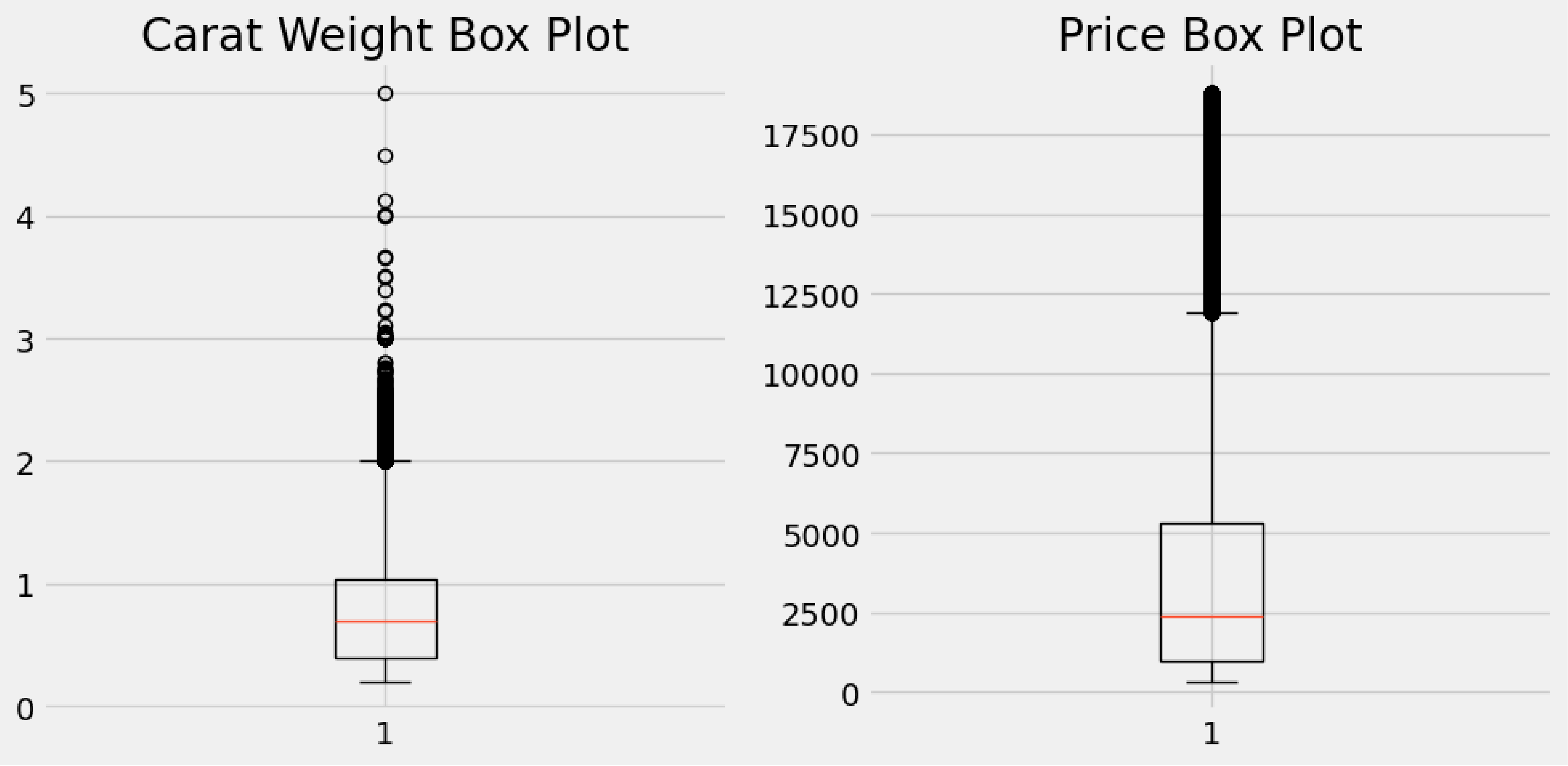
In [15]:
```
plt.figure(figsize=(6,5))
sns.scatterplot(x='carat', y='price', data=diamond_df, alpha=0.9)
plt.xlabel('Carat Weight')
plt.ylabel('Price')
plt.title('Carat Weight vs. Diamond Price')
plt.show()
```



In [14]:
```
plt.figure(figsize=(6, 5))
plt.hist(diamond_df['carat'], bins=10, edgecolor='k')
plt.xlabel('Carat Weight')
plt.ylabel('Frequency')
plt.title('Carat Weight Distribution')
plt.show()
```



In [18]:
```
plt.figure(figsize=(10,5))

plt.subplot(1, 2, 1)
plt.boxplot(diamond_df['carat'])
plt.title('Carat Weight Box Plot')

plt.subplot(1, 2, 2)
plt.boxplot(diamond_df['price'])
plt.title('Price Box Plot')

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

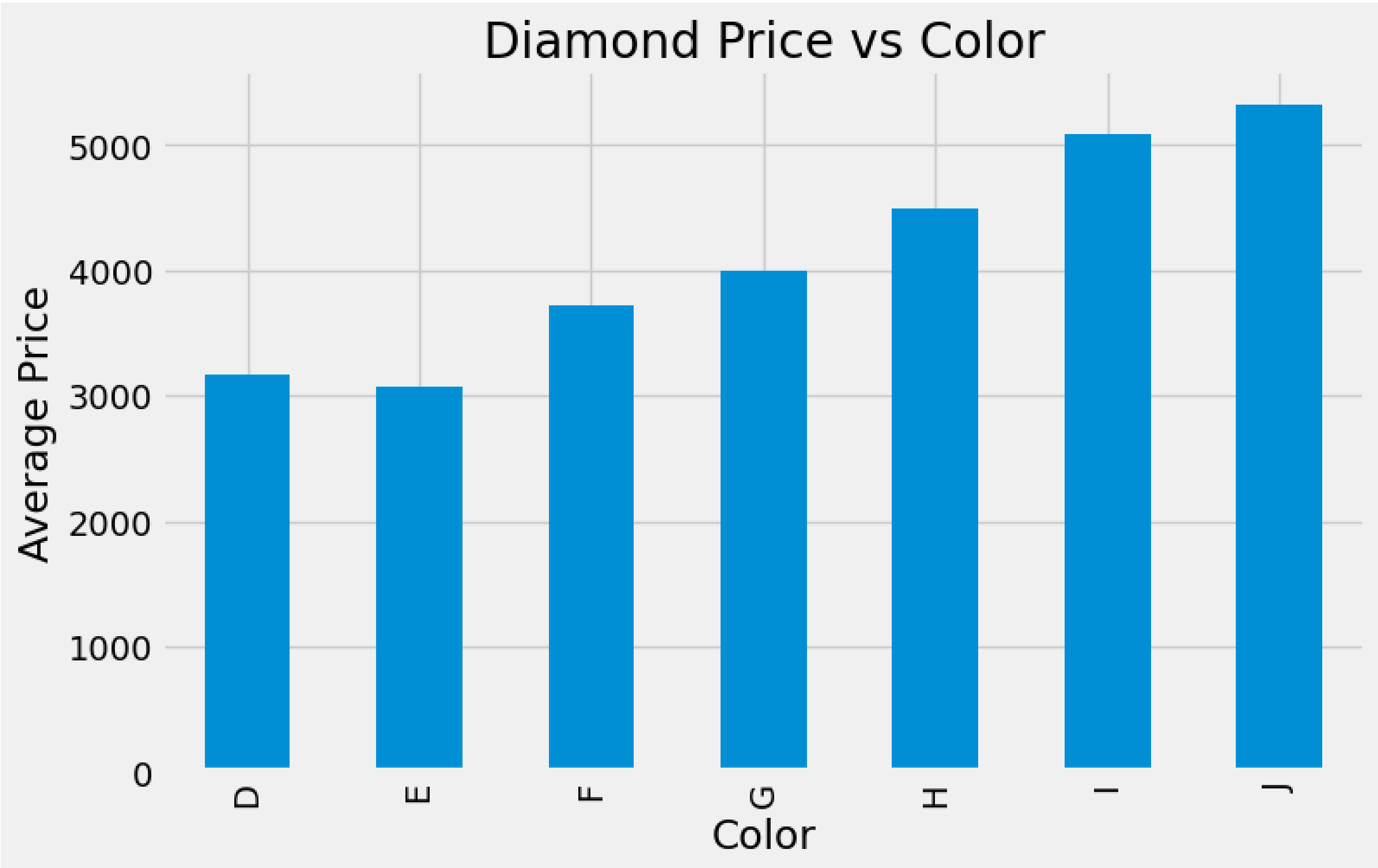


In [19]:
```
cut_quality_prices = diamond_df.groupby('cut')['price'].mean()
cut_quality_prices
```

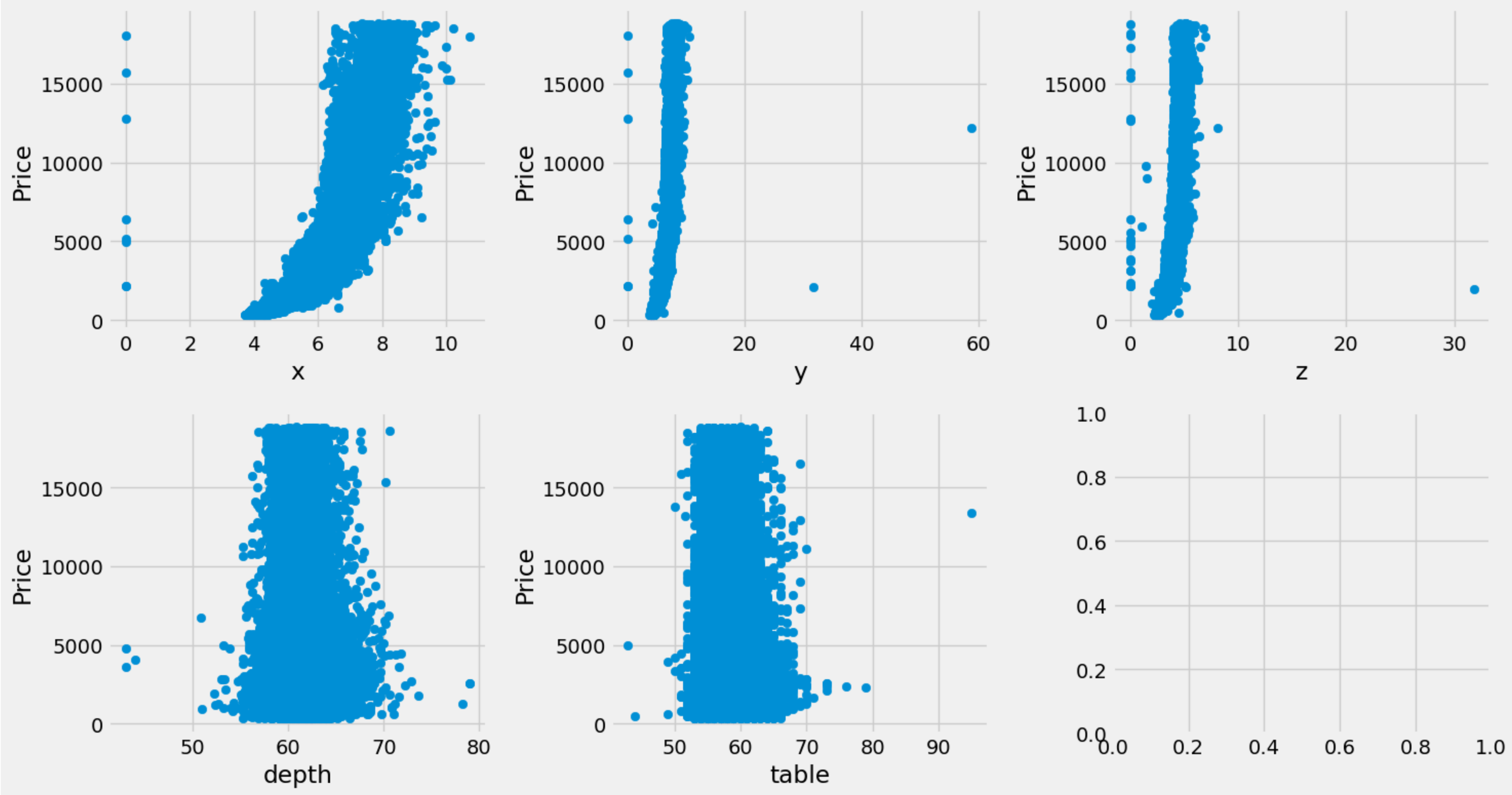
Out[19]:
```
cut
Fair         4358.757764
Good         3928.864452
Ideal        3457.541970
Premium      4584.257704
Very Good    3981.759891
Name: price, dtype: float64
```

In [20]:
```
plt.figure(figsize=(8,5))
cut_quality_prices.plot(kind='bar')
plt.xlabel('Cut Quality')
plt.ylabel('Average Price')
plt.title('Diamond Price vs. Cut Quality')
plt.show()
```

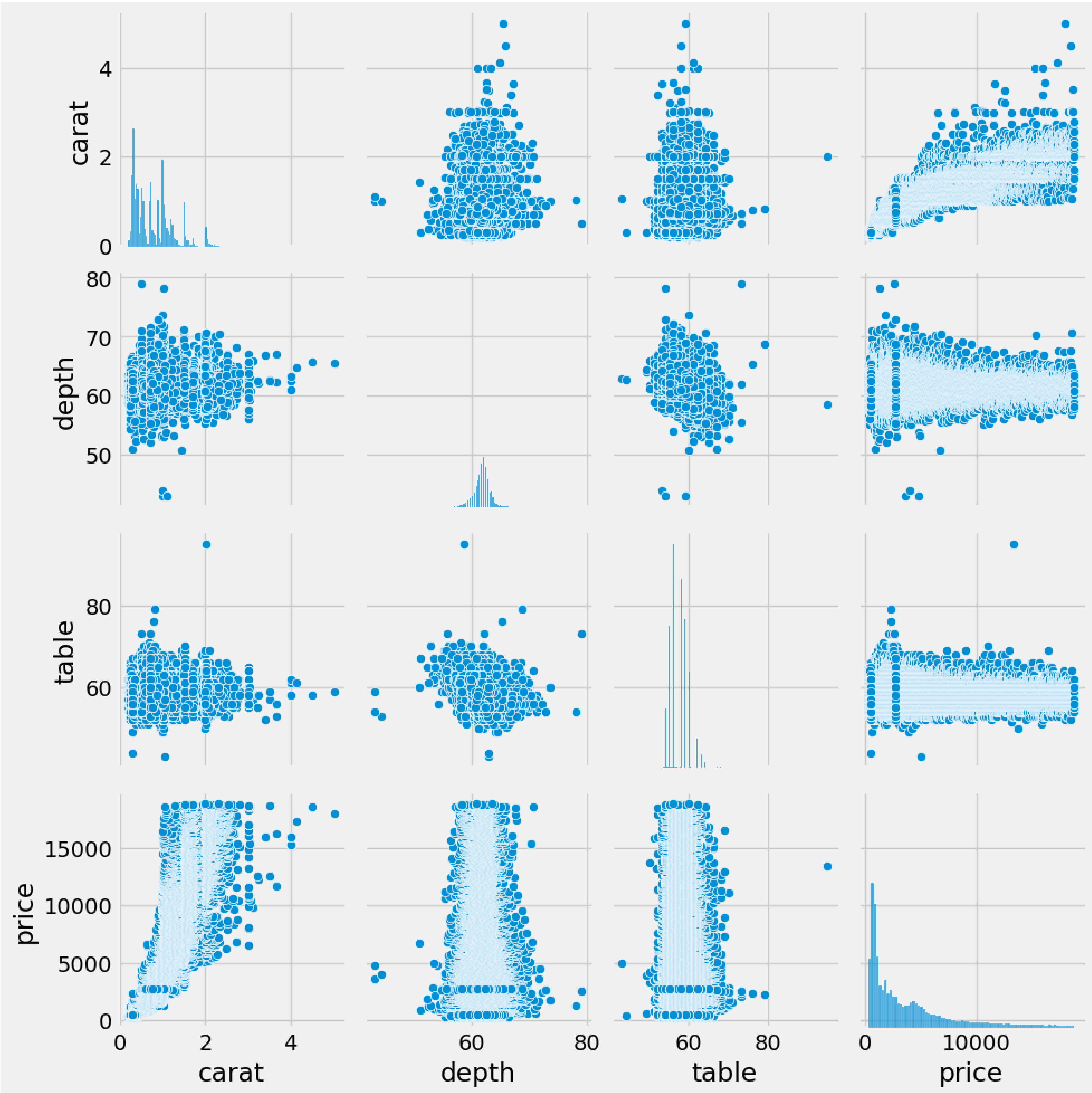
In [21]:
```
color_prices = diamond_df.groupby('color')['price'].mean()

plt.figure(figsize=(8,5))
color_prices.plot(kind='bar')
plt.xlabel('Color')
plt.ylabel('Average Price')
plt.title('Diamond Price vs Color')
plt.show()
```



In [22]:
```
fig, ax = plt.subplots(nrows=2, ncols=3, figsize=(15,8))
features = ['x', 'y', 'z', 'depth', 'table']

for i, feature in enumerate(features):
    row, col = divmod(i, 3)
    ax[row, col].scatter(diamond_df[feature], diamond_df['price'])
    ax[row, col].set_xlabel(feature)
    ax[row, col].set_ylabel('Price')

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



In [23]:
```
sns.pairplot(diamond_df[['carat', 'depth', 'table', 'price']])
plt.show()
```



In [24]:
```
numeric_features = ['carat', 'x', 'y', 'z', 'depth', 'table']

fig, ax = plt.subplots(2, 3, figsize=(15,7))
fig.suptitle('Regression Line Plots for Numeric Features vs. Price')

for i, feature in enumerate(numeric_features):
    row, col = divmod(i, 3)
    sns.regplot(x=feature, y='price', data=diamond_df, ax=ax[row, col])
    ax[row, col].set_xlabel(feature)
    ax[row, col].set_ylabel('Price')

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

In [25]:
```
X = diamond_df.drop(['price'], axis=1)
y = diamond_df['price']

# Split the data into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

print(X_train.shape, X_test.shape)
```

```
(43152, 9) (10788, 9)
```

In [26]:
```
label_encoder = LabelEncoder()

X_train['cut'] = label_encoder.fit_transform(X_train['cut'])
X_test['cut'] = label_encoder.transform(X_test['cut'])

X_train['color'] = label_encoder.fit_transform(X_train['color'])
X_test['color'] = label_encoder.transform(X_test['color'])

X_train['clarity'] = label_encoder.fit_transform(X_train['clarity'])
X_test['clarity'] = label_encoder.transform(X_test['clarity'])
```

In [27]:
```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [29]:
```
from sklearn.preprocessing import MinMaxScaler

scaler= MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [30]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)


y_pred = lin_reg.predict(X_test_scaled)


mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'Mean Absolute Error: {mae:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'R2 Score: {r2:.2f}')
```

```
Mean Squared Error: 1790036.03
Mean Absolute Error: 859.18
Root Mean Squared Error: 1337.92
R2 Score: 0.89
```

In [31]:
```python
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train_scaled, y_train)


y_pred = tree_model.predict(X_test_scaled)


mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'Mean Absolute Error: {mae:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'R2 Score: {r2:.2f}')
```

```
Mean Squared Error: 544648.42
Mean Absolute Error: 359.92
Root Mean Squared Error: 738.00
R2 Score: 0.97
```

In [32]:
```python
xgb_model = xgb.XGBRFRegressor(objective='reg:squarederror', random_state=42)
xgb_model.fit(X_train_scaled, y_train)
y_pred = xgb_model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'Mean Absolute Error: {mae:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'R2 Score: {r2:.2f}')
```

```
Mean Squared Error: 765127.78
Mean Absolute Error: 465.60
Root Mean Squared Error: 874.72
R2 Score: 0.95
```

In [ ]: