In [1]:

```python
# ==== Package Check & Auto-Install ==========================================
import importlib
import sys
import subprocess

def install_if_missing(pkg_name, import_name=None):
    import_name = import_name or pkg_name
    try:
        importlib.import_module(import_name)
        print(f"✓ {pkg_name} already installed")
    except ImportError:
        print(f"⚠ {pkg_name} not found — installing…")
        subprocess.check_call([sys.executable, "-m", "pip", "install", pkg_name])
        print(f"✔ Installed {pkg_name} successfully")

required_packages = [
    ("matplotlib", "matplotlib"),
    ("numpy", "numpy"),
    ("pandas", "pandas"),
    ("seaborn", "seaborn"),
    ("statsmodels", "statsmodels"),
    ("scikit-learn", "sklearn"),
    ("category_encoders", "category_encoders")
]

for pkg, import_name in required_packages:
    install_if_missing(pkg, import_name)

# ==== Imports ==============================================================

# hide warnings to keep printout clean
import warnings
warnings.filterwarnings("ignore")

import os
os.environ["PANDAS_IGNORE_BOTTLENECK"] = "1"
import numpy as np
import pandas as pd
import seaborn as sns
import math
```

```python
import statsmodels.formula.api as smf
from category_encoders import TargetEncoder
import matplotlib.pyplot as plt

# sklearn
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import train_test_split, KFold, cross_val_score, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
```

```
✓ matplotlib already installed
✓ numpy already installed
✓ pandas already installed
✓ seaborn already installed
✓ statsmodels already installed
✓ scikit-learn already installed
✓ category_encoders already installed
```

Data Cleaning

In [2]:

```python
df = pd.read_csv('data/rideshare_kaggle.csv')

# select variables needed for this project
selected_vars = [
    "id", "price", "distance", "cab_type", "name",
    "timestamp", "hour", "day", "source", "destination",
    "temperature", "precipIntensity", "precipProbability", "cloudCover",
    "surge_multiplier", "short_summary"
]

df = df[selected_vars].copy()
df.head()
```

Out[2]:

	id	price	distance	cab_type	name	timestamp	hour	day	source	destination	temperature	pre
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	5.0	0.44	Lyft	Shared	1.544953e+09	9	16	Haymarket Square	North Station	42.34	
1	4bd23055-6827-41c6-b23b-3c491f24e74d	11.0	0.44	Lyft	Lux	1.543284e+09	2	27	Haymarket Square	North Station	43.58	
2	981a3613-77af-4620-a42a-0c0866077d1e	7.0	0.44	Lyft	Lyft	1.543367e+09	1	28	Haymarket Square	North Station	38.33	
3	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	26.0	0.44	Lyft	Lux Black XL	1.543554e+09	4	30	Haymarket Square	North Station	34.38	
4	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	9.0	0.44	Lyft	Lyft XL	1.543463e+09	3	29	Haymarket Square	North Station	37.44	

In [3]:

```
# rename the original day to day_of_month
df['day_of_month'] = df['day'].astype(int)

# fix timestamp and create proper day variables
df['datetime'] = pd.to_datetime(df['timestamp'], unit='s')

df["month_name"] = df["datetime"].dt.month_name()  # January, February, ...
df["day_of_week"] = df["datetime"].dt.day_name()  # Monday, Tuesday, ...


# create a new binary indicator for is_weekend
df['is_weekend'] = df['day_of_week'].isin(["Saturday", "Sunday"])

# remove variables not needed
df = df.drop(columns=["day", "timestamp", "datetime"])
```

```
df.head()
```

Out[3]:

	id	price	distance	cab_type	name	hour	source	destination	temperature	precipIntensity	precipP
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	5.0	0.44	Lyft	Shared	9	Haymarket Square	North Station	42.34	0.0000	
1	4bd23055-6827-41c6-b23b-3c491f24e74d	11.0	0.44	Lyft	Lux	2	Haymarket Square	North Station	43.58	0.1299	
2	981a3613-77af-4620-a42a-0c0866077d1e	7.0	0.44	Lyft	Lyft	1	Haymarket Square	North Station	38.33	0.0000	
3	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	26.0	0.44	Lyft	Lux Black XL	4	Haymarket Square	North Station	34.38	0.0000	
4	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	9.0	0.44	Lyft	Lyft XL	3	Haymarket Square	North Station	37.44	0.0000	

In [4]:

```
# General info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 693071 entries, 0 to 693070
Data columns (total 18 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   id                 693071 non-null  object 
 1   price              637976 non-null  float64
 2   distance           693071 non-null  float64
 3   cab_type           693071 non-null  object 
 4   name               693071 non-null  object 
 5   hour               693071 non-null  int64  
 6   source             693071 non-null  object 
 7   destination        693071 non-null  object 
 8   temperature        693071 non-null  float64
 9   precipIntensity    693071 non-null  float64
 10  precipProbability  693071 non-null  float64
 11  cloudCover         693071 non-null  float64
 12  surge_multiplier   693071 non-null  float64
 13  short_summary      693071 non-null  object 
 14  day_of_month       693071 non-null  int64  
 15  month_name         693071 non-null  object 
 16  day_of_week        693071 non-null  object 
 17  is_weekend         693071 non-null  bool   
dtypes: bool(1), float64(7), int64(2), object(8)
memory usage: 90.6+ MB
```

In [5]:

```
# Shape
print(f"Rows: {df.shape[0]}, Columns: {df.shape[1]}")
```

```
Rows: 693071, Columns: 18
```

In [6]:

```
# Check for duplicates
print(f"Duplicated rows: {df.duplicated().sum()}")
```

```
Duplicated rows: 0
```

In [7]:

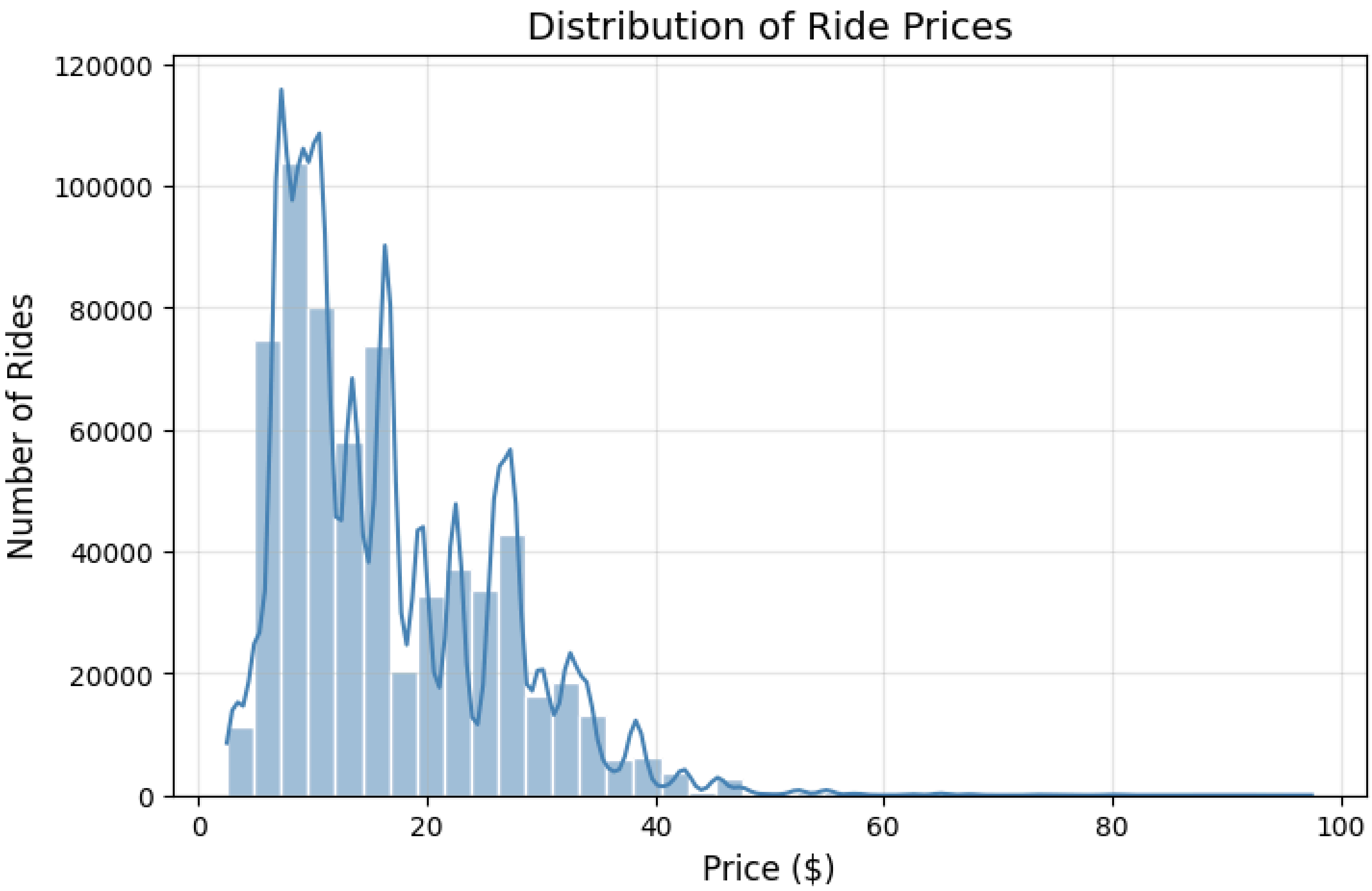
```
# number of missing values
df.isna().sum().sort_values(ascending=False).head()
```

Out[7]:
```
price           55095
id                  0
day_of_week         0
month_name          0
day_of_month        0
dtype: int64
```

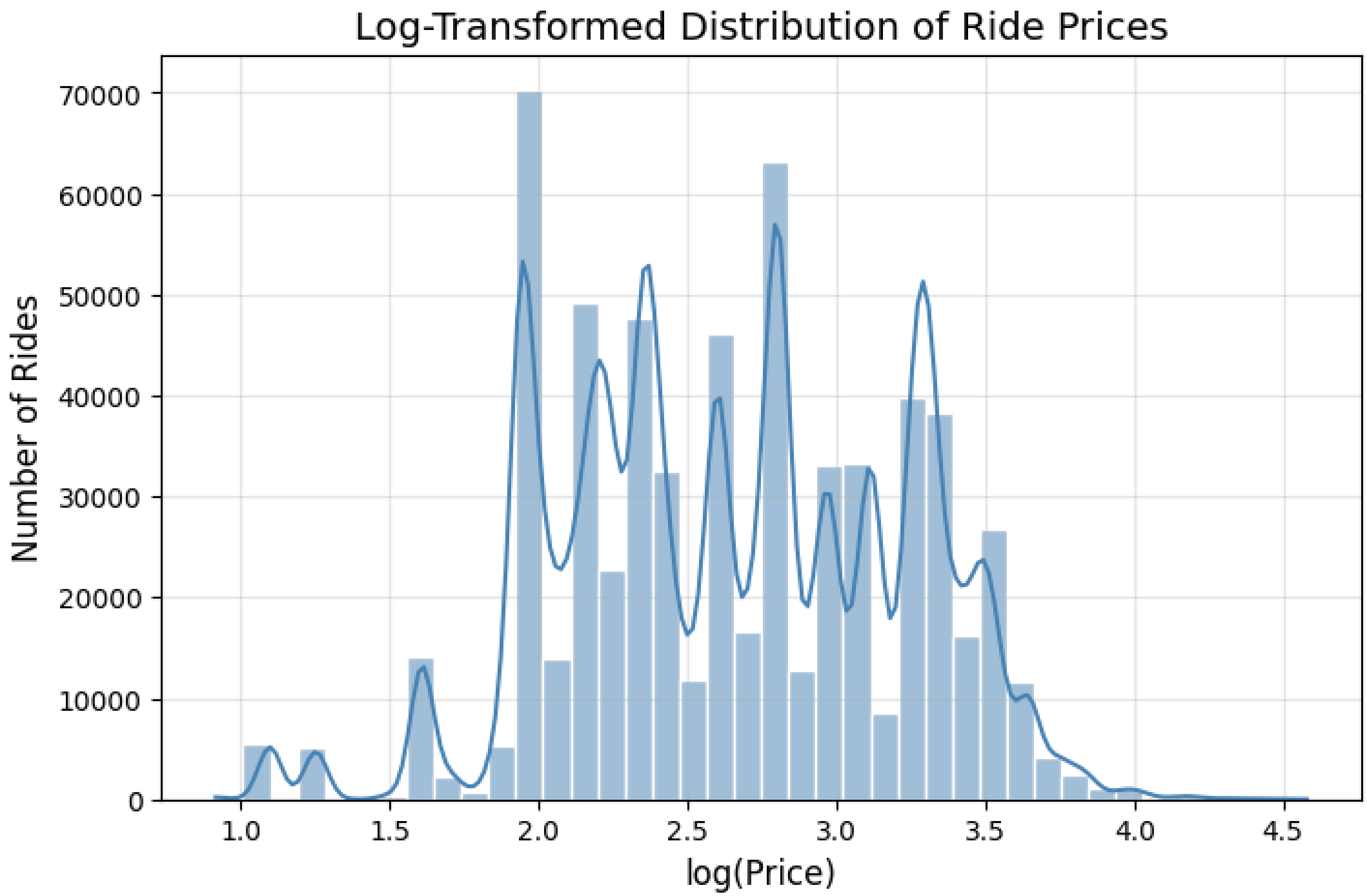
EDA

Outcome Variable Check

In [8]:
```
# distribution of price (original), with na values removed
prices = df['price'].dropna()

plt.figure(figsize=(8,5))
sns.histplot(prices, bins=40, kde=True, color='steelblue', edgecolor='white')

plt.title('Distribution of Ride Prices', fontsize=14)
plt.xlabel('Price ($)', fontsize=12)
plt.ylabel('Number of Rides', fontsize=12)
plt.grid(alpha=0.3)
plt.show()
```



In [9]:
```
# distribution of price (transformed), with na values removed
# Use np.log1p() to handle any zero or near-zero prices safely
prices = np.log(df['price'].dropna())

plt.figure(figsize=(8,5))
sns.histplot(prices, bins=40, kde=True, color='steelblue', edgecolor='white')

plt.title('Log-Transformed Distribution of Ride Prices', fontsize=14)
plt.xlabel('log(Price)', fontsize=12)
plt.ylabel('Number of Rides', fontsize=12)
plt.grid(alpha=0.3)
plt.show()
```



Categorical Variables Check

In [10]:
```
categorical_var = [
    "cab_type",
    "name",
    "source",
    "destination",
    "short_summary",
    "month_name",
    "day_of_week",
```

```
    "is_weekend"
]
```

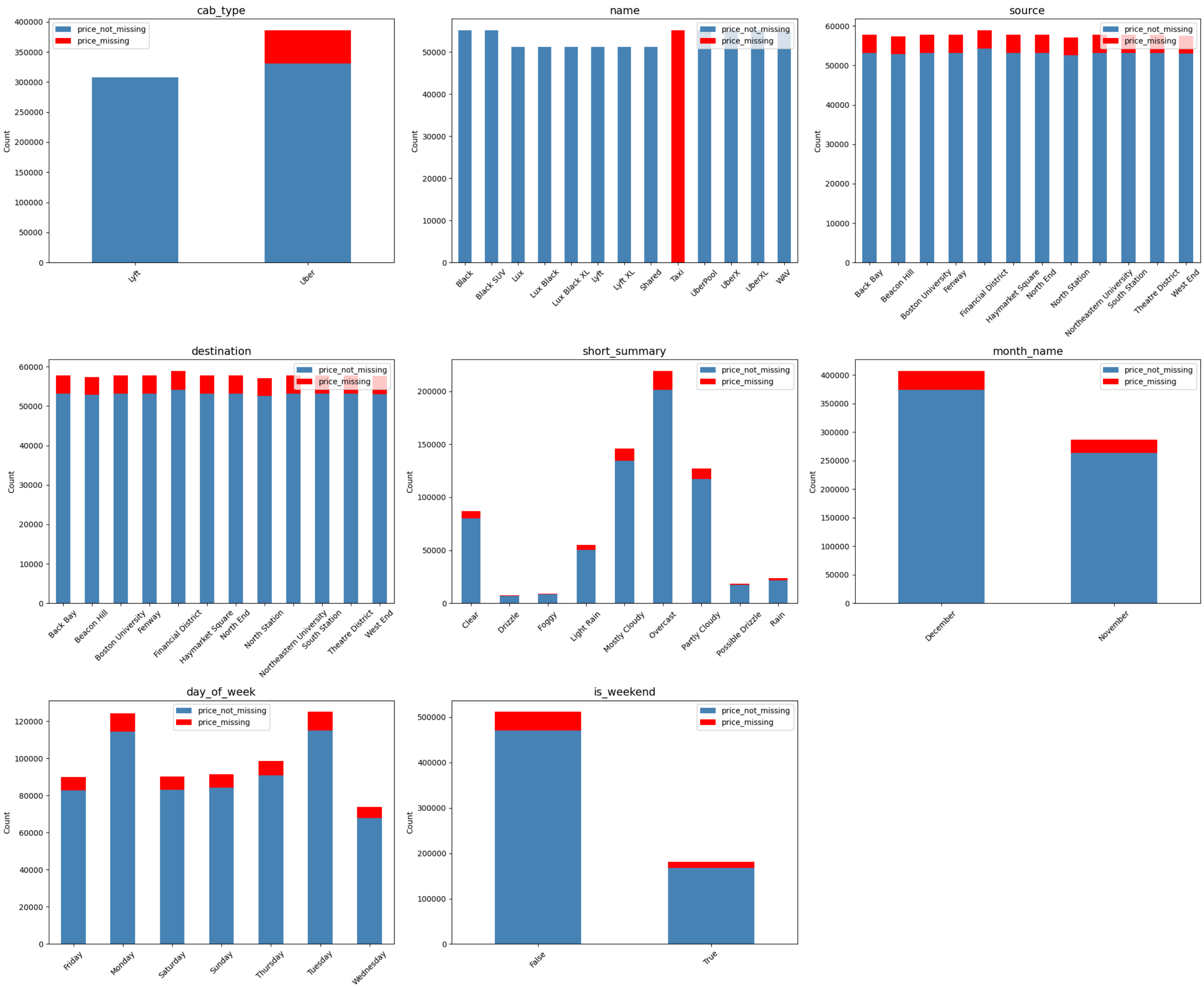
In [11]:
```
cat_summary = df[categorical_var].describe()

print("\n===== Categorical Variables Summary =====")
display(cat_summary)
```

```
===== Categorical Variables Summary =====
```

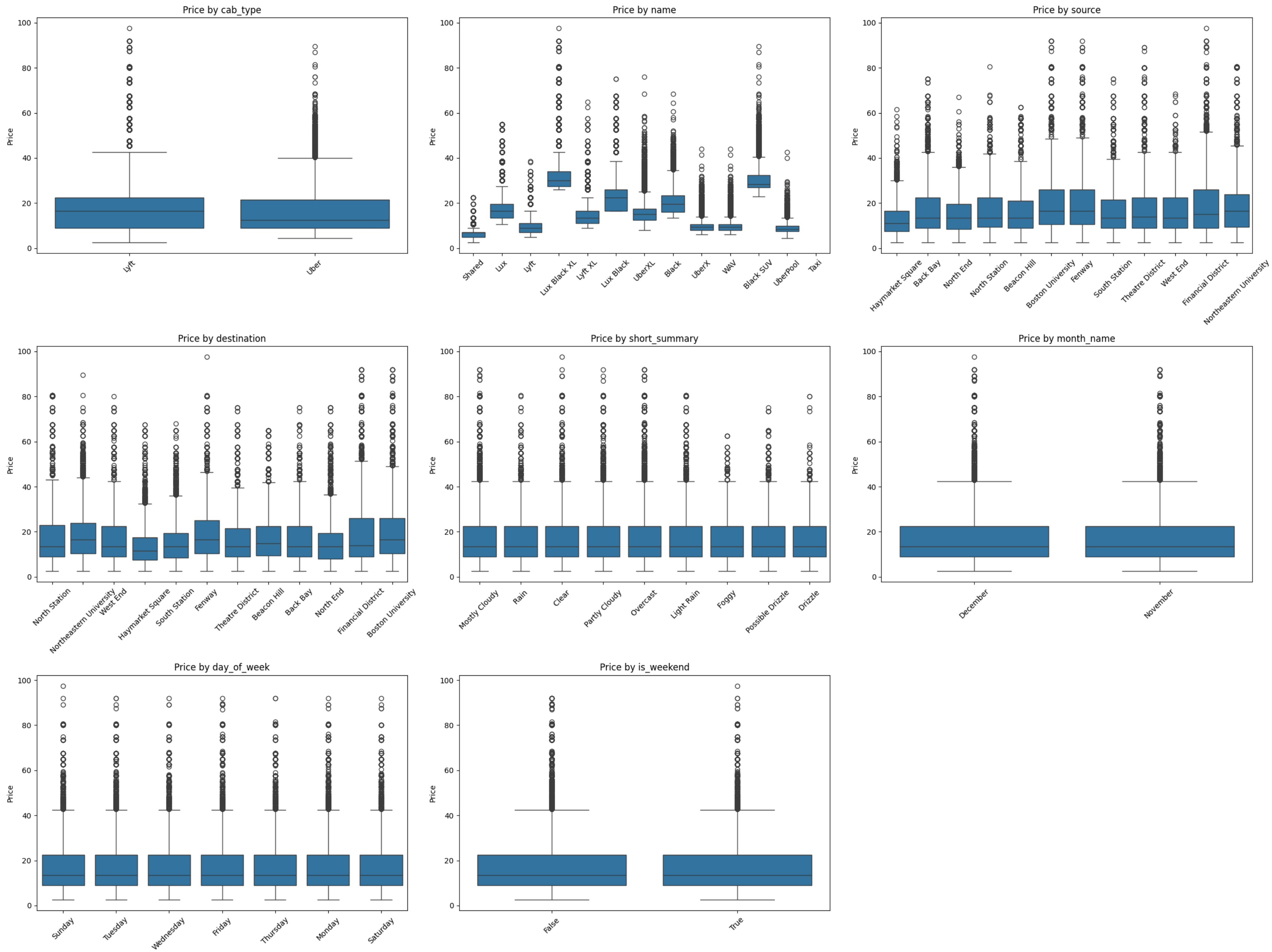
	cab_type	name	source	destination	short_summary	month_name	day_of_week	is_weekend
count	693071	693071	693071	693071	693071	693071	693071	693071
unique	2	13	12	12	9	2	7	2
top	Uber	UberXL	Financial District	Financial District	Overcast	December	Tuesday	False
freq	385663	55096	58857	58851	218895	406614	124949	511373

In [12]:
```
n = len(categorical_var)
rows = math.ceil(n / 3)

fig, axes = plt.subplots(rows, 3, figsize=(22, 6 * rows))
axes = axes.flatten()

for i, col in enumerate(categorical_var):
    # group category by price missing / not missing
    counts = df.groupby([col, df['price'].isna()]).size().unstack(fill_value=0)
    # rename columns for clarity
    counts.columns = ['price_not_missing', 'price_missing']

    # stacked bar plot
    counts[['price_not_missing', 'price_missing']].plot(
        kind='bar',
        stacked=True,
        ax=axes[i],
        color=['steelblue', 'red']  # normal + missing price in red
    )

    axes[i].set_title(f"{col}", fontsize=14)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("Count")
```

```
    axes[i].tick_params(axis='x', labelrotation=45)

# hide unused subplots if number of vars % 3 != 0
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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


cab_type
price_not_missing
price_missing
Count
Lyft
Uber
name
Black
Black SUV
Lux
Lux Black
Lux Black XL
Lyft
Lyft XL
Shared
Taxi
UberPool
UberX
UberXL
WAV
source
Back Bay
Beacon Hill
Boston University
Fenway
Financial District
Haymarket Square
North End
North Station
Northeastern University
South Station
Theatre District
West End
destination
short_summary
Clear
Drizzle
Foggy
Light Rain
Mostly Cloudy
Overcast
Partly Cloudy
Possible Drizzle
Rain
month_name
December
November
day_of_week
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
is_weekend
False
True

In [13]:
```
n = len(categorical_var)
rows = math.ceil(n / 3)

fig, axes = plt.subplots(rows, 3, figsize=(24, 6 * rows))
axes = axes.flatten()

for i, col in enumerate(categorical_var):
    sns.boxplot(data=df, x=col, y="price", ax=axes[i])
    axes[i].set_title(f"Price by {col}")
    axes[i].set_xlabel("")
    axes[i].set_ylabel("Price")
    axes[i].tick_params(axis="x", rotation=45)

# hide unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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



Across categorical predictors, price exhibits relatively consistent patterns with a few meaningful differences. Cab type and service name show the strongest price variation, with premium services (e.g., Lyft Lux, Lyft XL) generally yielding higher fares compared to standard offerings. Pickup and drop-off locations also influence pricing, with certain hubs (e.g., airports and major transport stations) associated with higher median fares, likely reflecting longer distances and greater demand. In contrast, weather-related categories (short_summary), day of week, weekend indicator, and month show only modest differences in price distributions, suggesting they play a secondary role relative to service level and location. Overall, the categorical EDA indicates that ride type and geographic endpoints are key drivers of price, while temporal and weather-based variables may have weaker direct effects.

Numerical Variables Check

In [14]:
```
numerical_var = [
    "distance",
    "hour",
    "temperature",
    "precipIntensity",
    "precipProbability",
    "cloudCover",
    "surge_multiplier",
    "day_of_month"
]
```

In [15]:
```
num_summary = df[numerical_var].describe().round(2)

print("\n===== Numerical Variables Summary =====")
display(num_summary)
```

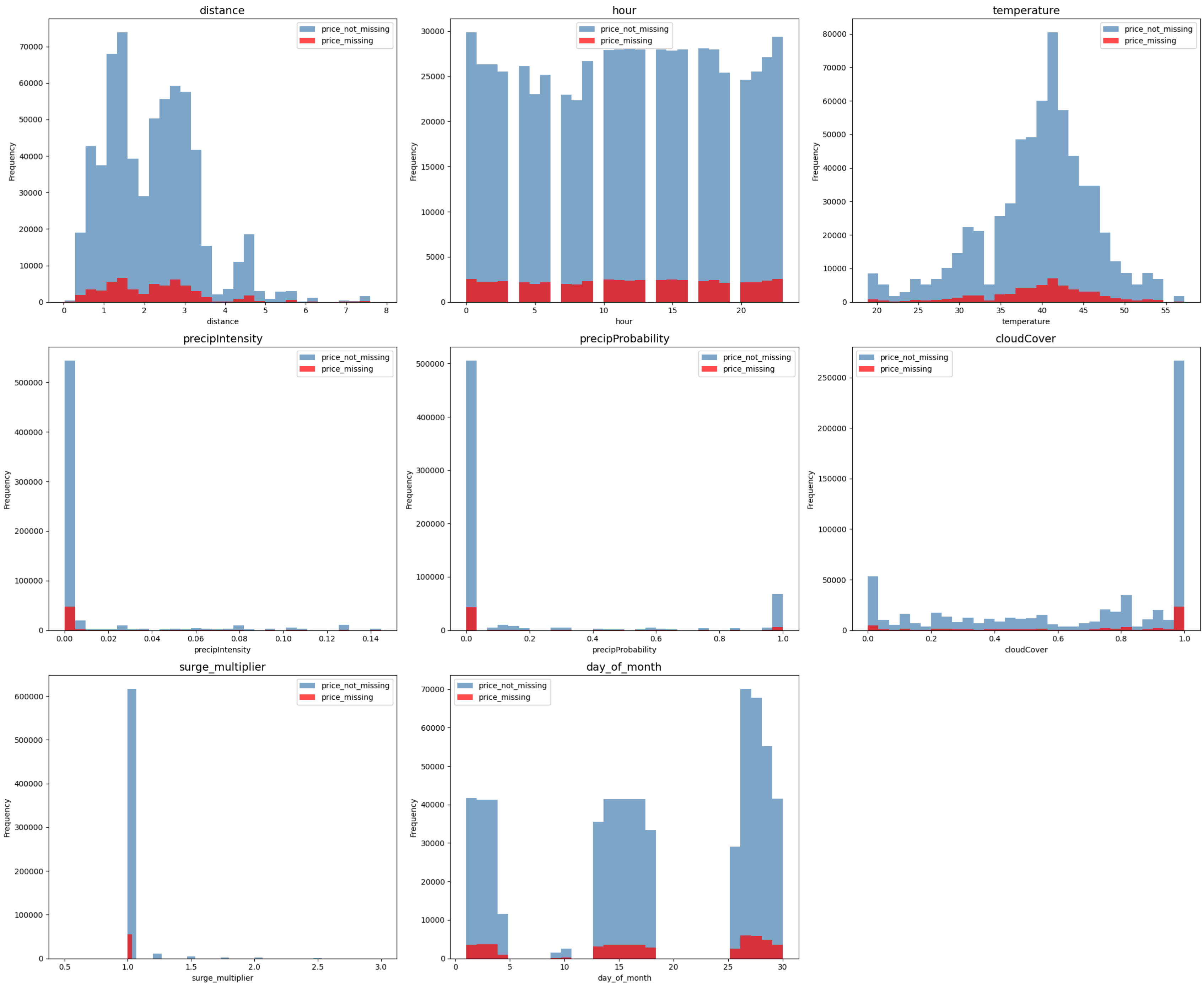
```
===== Numerical Variables Summary =====
```

	distance	hour	temperature	precipIntensity	precipProbability	cloudCover	surge_multiplier	day_of_month
count	693071.00	693071.00	693071.00	693071.00	693071.00	693071.00	693071.00	693071.00
mean	2.19	11.62	39.58	0.01	0.15	0.69	1.01	17.79
std	1.14	6.95	6.73	0.03	0.33	0.36	0.09	9.98
min	0.02	0.00	18.91	0.00	0.00	0.00	1.00	1.00
25%	1.28	6.00	36.45	0.00	0.00	0.37	1.00	13.00
50%	2.16	12.00	40.49	0.00	0.00	0.82	1.00	17.00
75%	2.92	18.00	43.58	0.00	0.00	1.00	1.00	28.00
max	7.86	23.00	57.22	0.14	1.00	1.00	3.00	30.00

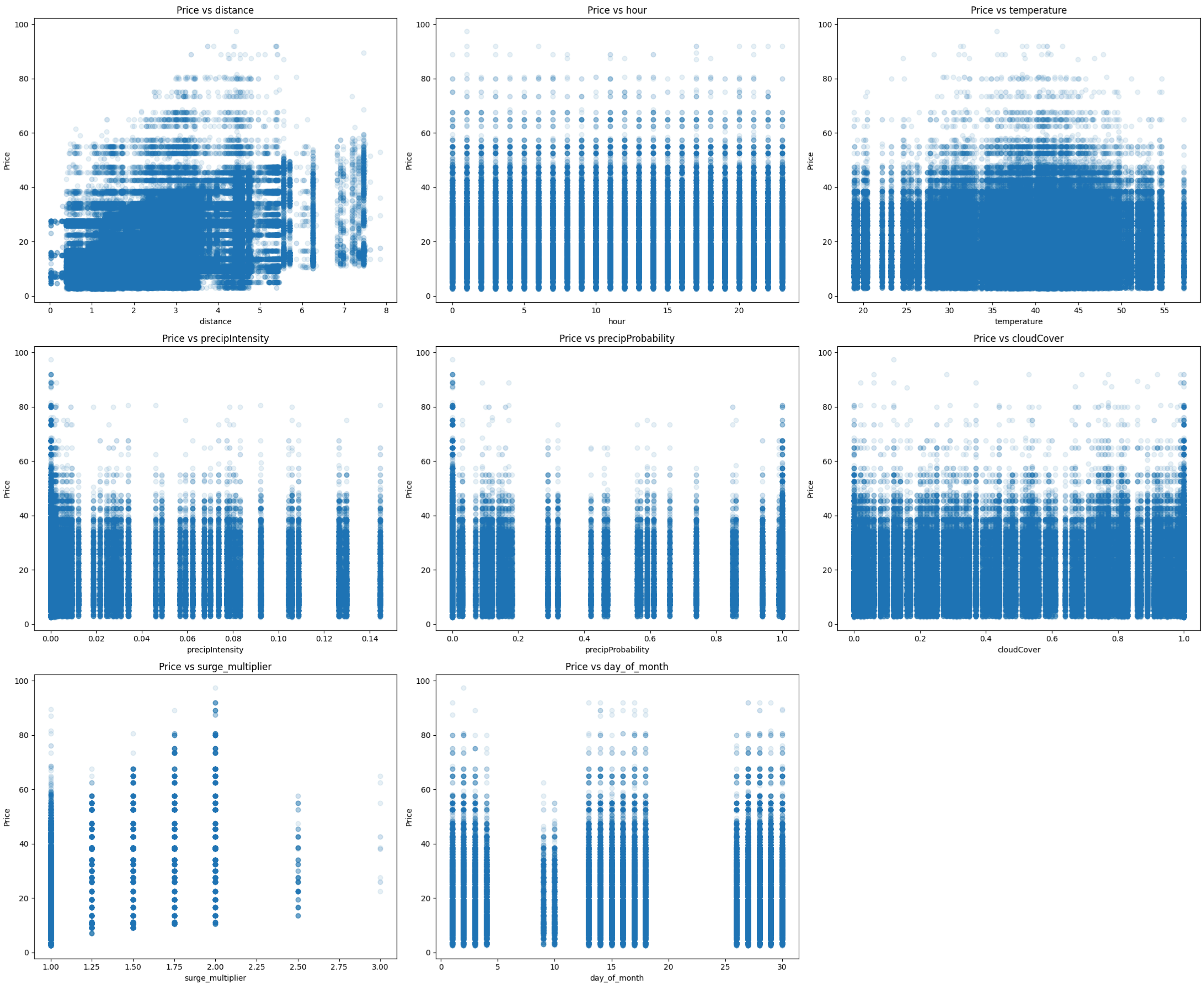
In [16]:

```
n = len(numerical_var)
rows = math.ceil(n / 3)

fig, axes = plt.subplots(rows, 3, figsize=(22, 6 * rows))
axes = axes.flatten()

for i, col in enumerate(numerical_var):
    # valid price values
    df[df["price"].notna()][col].plot(
        kind='hist',
        bins=30,
        alpha=0.7,
        color='steelblue',
        ax=axes[i],
        label="price_not_missing"
    )

    # missing price values
    df[df["price"].isna()][col].plot(
        kind='hist',
        bins=30,
        alpha=0.7,
        color='red',
        ax=axes[i],
        label="price_missing"
```

```
    )

    axes[i].set_title(f"{col}", fontsize=14)
    axes[i].set_xlabel(col)
    axes[i].set_ylabel("Frequency")
    axes[i].legend()

# hide unused subplots (if number of vars not divisible by 3)
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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


distance
price_not_missing
price_missing
Frequency
distance
hour
hour
temperature
temperature
precipIntensity
precipIntensity
precipProbability
precipProbability
cloudCover
cloudCover
surge_multiplier
surge_multiplier
day_of_month
day_of_month

In [17]:
```
n = len(numerical_var)
rows = math.ceil(n / 3)

fig, axes = plt.subplots(rows, 3, figsize=(22, 6 * rows))
axes = axes.flatten()

for i, col in enumerate(numerical_var):
    axes[i].scatter(df[col], df["price"], alpha=0.1)
    axes[i].set_title(f"Price vs {col}")
    axes[i].set_xlabel(col)
    axes[i].set_ylabel("Price")

# hide unused subplots if not multiple of 3
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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


Price vs distance
Price
distance
Price vs hour
Price
hour
Price vs temperature
Price
temperature
Price vs precipIntensity
Price
precipIntensity
Price vs precipProbability
Price
precipProbability
Price vs cloudCover
Price
cloudCover
Price vs surge_multiplier
Price
surge_multiplier
Price vs day_of_month
Price
day_of_month

In [18]:
```
# Outlier detection
outlier_summary = {}

for col in numerical_var:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    outliers = ((df[col] < lower) | (df[col] > upper)).sum()
    outlier_summary[col] = outliers

pd.DataFrame.from_dict(outlier_summary, orient='index', columns=['Outlier Count'])
```

Out[18]:

	Outlier Count
distance	8662
hour	0
temperature	36659
precipIntensity	150828
precipProbability	150828
cloudCover	0
surge_multiplier	20975
day_of_month	0

In [19]:
```
# Scalling
scaler = StandardScaler()
scaled = scaler.fit_transform(df[numerical_var])

scaled_df = pd.DataFrame(scaled, columns=[c + '_scaled' for c in numerical_var])
scaled_df.head()
```

Out[19]:

	distance_scaled	hour_scaled	temperature_scaled	precipIntensity_scaled	precipProbability_scaled	cloudCover_scaled
0	-1.536021	-0.376957	0.409691	-0.331672	-0.444083	0.093430
1	-1.536021	-1.384425	0.594048	4.497229	2.596003	0.874388
2	-1.536021	-1.528349	-0.186496	-0.331672	-0.444083	-1.831073
3	-1.536021	-1.096577	-0.773763	-0.331672	-0.444083	-1.914747
4	-1.536021	-1.240501	-0.318817	-0.331672	-0.444083	-0.687527

Findings: The numerical predictors in our dataset exhibit a wide range of distributional behaviors.

- Distance shows a multimodal distribution reflecting different ride types (short local trips vs. longer airport rides).
- Hour displays usage across all 24 hours, with moderately higher demand during daytime and early evening.
- CloudCover shows a bimodal distribution, heavily concentrated at both 0 (clear) and 1 (fully overcast), with fewer values in between.
- Temperature is moderately distributed.
- Weather variables, especially precipIntensity and precipProbability, are extremely right-skewed, with over 150,000 outliers each, reflecting the rarity of heavy precipitation events. CloudCover is left-skewed.
- Surge_multiplier is highly concentrated at 1.0, indicating that the vast majority of rides occur without surge pricing.
- Day_of_month has three peaks, suggesting limited temporal coverage in the dataset.

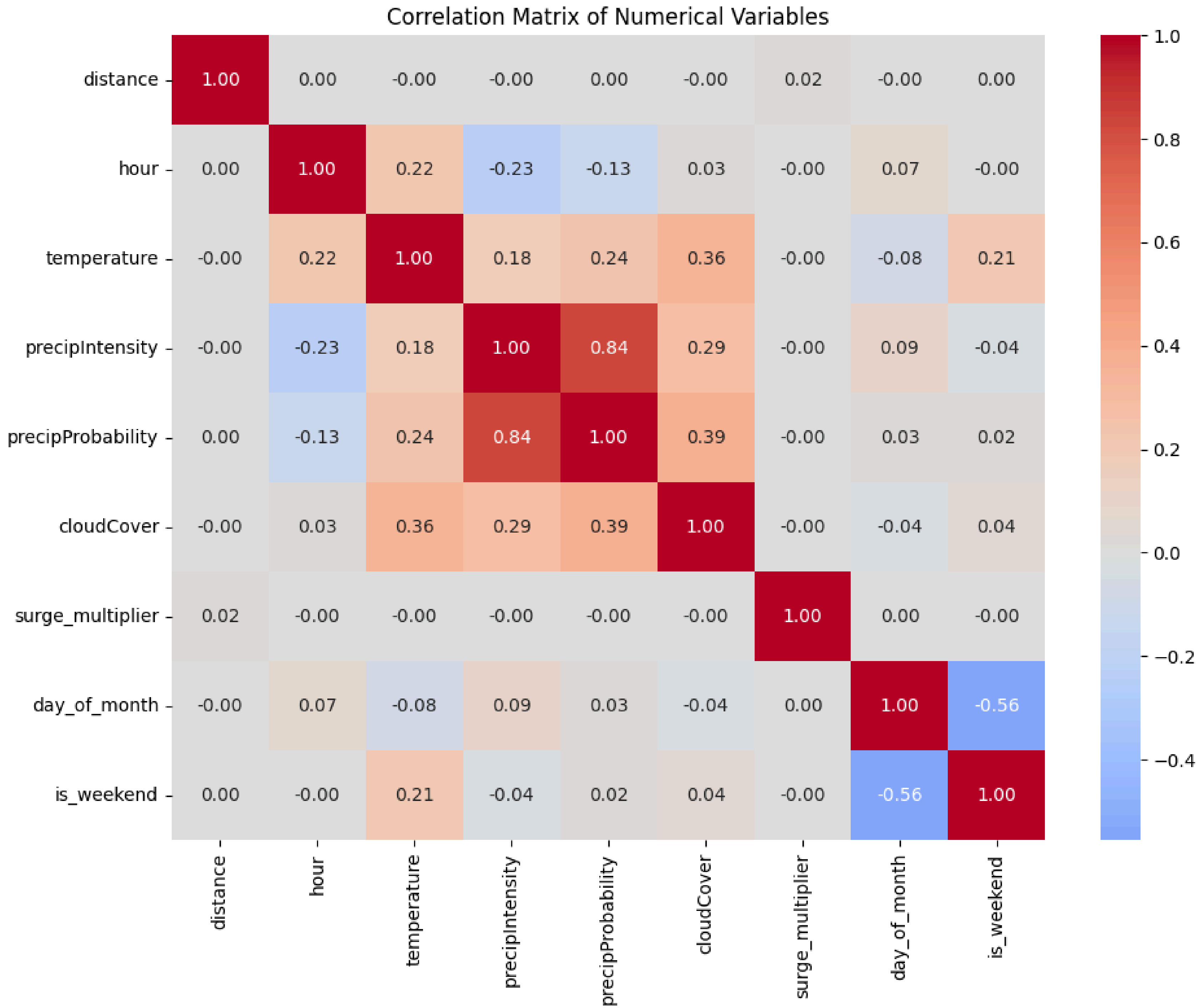
Scatterplots further confirm that distance is the only predictor showing a visible positive relationship with price. Other variables exhibit weak or no linear associations.

All predictors were standardized.

Correlation check

In [20]:
```
corr_var = [
    "distance",
    "hour",
    "temperature",
    "precipIntensity",
    "precipProbability",
```

```
    "cloudCover",
    "surge_multiplier",
    "day_of_month",
    "is_weekend"
]

corr_matrix = df[corr_var].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', center=0)
plt.title('Correlation Matrix of Numerical Variables')
plt.tight_layout()
plt.show()

print("Correlation Matrix:")
print(corr_matrix)
```


Correlation Matrix of Numerical Variables
distance
hour
temperature
precipIntensity
precipProbability
cloudCover
surge_multiplier
day_of_month
is_weekend
1.00 0.00 -0.00 -0.00 0.00 -0.00 0.02 -0.00 0.00
0.00 1.00 0.22 -0.23 -0.13 0.03 -0.00 0.07 -0.00
-0.00 0.22 1.00 0.18 0.24 0.36 -0.00 -0.08 0.21
-0.00 -0.23 0.18 1.00 0.84 0.29 -0.00 0.09 -0.04
0.00 -0.13 0.24 0.84 1.00 0.39 -0.00 0.03 0.02
-0.00 0.03 0.36 0.29 0.39 1.00 -0.00 -0.04 0.04
0.02 -0.00 -0.00 -0.00 -0.00 -0.00 1.00 0.00 -0.00
-0.00 0.07 -0.08 0.09 0.03 -0.04 0.00 1.00 -0.56
0.00 -0.00 0.21 -0.04 0.02 0.04 -0.00 -0.56 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

```
Correlation Matrix:
                   distance      hour  temperature  precipIntensity  \
distance           1.000000  0.002280    -0.002884        -0.000256   
hour               0.002280  1.000000     0.218769        -0.233349   
temperature       -0.002884  0.218769     1.000000         0.182724   
precipIntensity   -0.000256 -0.233349     0.182724         1.000000   
precipProbability  0.000371 -0.129725     0.239165         0.838470   
cloudCover        -0.000905  0.026525     0.355156         0.288960   
surge_multiplier   0.024769 -0.000077    -0.001572        -0.001530   
day_of_month      -0.000627  0.066090    -0.084116         0.091664   
is_weekend         0.001523 -0.002531     0.212279        -0.037736   

                   precipProbability  cloudCover  surge_multiplier  \
distance                    0.000371   -0.000905          0.024769   
hour                       -0.129725    0.026525         -0.000077   
temperature                 0.239165    0.355156         -0.001572   
precipIntensity             0.838470    0.288960         -0.001530   
precipProbability           1.000000    0.387114         -0.002769   
cloudCover                  0.387114    1.000000         -0.002103   
surge_multiplier           -0.002769   -0.002103          1.000000   
day_of_month                0.030903   -0.041190          0.001389   
is_weekend                  0.024864    0.039040         -0.001320   

                   day_of_month  is_weekend  
distance              -0.000627    0.001523  
hour                   0.066090   -0.002531  
temperature           -0.084116    0.212279  
precipIntensity        0.091664   -0.037736  
precipProbability      0.030903    0.024864  
cloudCover            -0.041190    0.039040  
surge_multiplier       0.001389   -0.001320  
day_of_month           1.000000   -0.555498  
is_weekend            -0.555498    1.000000  
```

Insights:

- `precipIntensity` and `precipProbability` are highly correlated (r = 0.84).
  This is reasonable because higher rainfall intensity typically occurs when the probability of rain is also high—both variables capture related aspects of precipitation events.
  We will drop `precipProbability` and use `precipIntensity` for modeling.

- `temperature` and `cloudCover` show moderate correlation (r = 0.36).
  This is reasonable because cloud cover influences temperature patterns—clouds trap heat at night and reduce warming during the day—creating a natural meteorological relationship.
  We will drop `cloudCover` and use `temperature` for modeling.

- `name` and `cab_type` are perfectly correlated (r = 1).
  This is expected because `name` (the specific ride type) is nested within `cab_type` (Uber or Lyft), so both variables encode the same information.
  We will drop `name` and use `cab_type` for modeling.

- `is_weekend` and `day_of_month` are negatively correlated (r = -0.56). This is expected because `is_weekend` is a binary indicator created from `day_of_month`. We will drop `day_of_month` and use `is_weekend` for modeling.

- Most remaining variables show weak correlations ($|r| < 0.4$), indicating minimal multicollinearity concerns among the retained predictors.

final features selection: feature_cols = ['distance', 'surge_multiplier', 'temperature', 'precipIntensity', 'hour', 'is_weekend', 'month_name', 'source', 'destination', 'name', 'cab_type']

Baseline model: log(price) ~ distance

In [21]:
```
# train-test-validation split

# remove missing values
df_clean = df.dropna()

X = df_clean[['distance']].values
y = np.log(df_clean['price']) # log-transform price

# first split: 80% train+val, 20% test
X_train_and_val, X_test, y_train_and_val, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# second split: 75% of train+val for train (60% overall), 25% for val (20% overall)
X_train, X_val, y_train, y_val = train_test_split(
```

```
    X_train_and_val, y_train_and_val, test_size=0.25, random_state=42
)

print(f"\nData split sizes:")
print(f"Train: {len(X_train)} ({len(X_train)/len(df_clean)*100:.1f}%)")
print(f"Validation: {len(X_val)} ({len(X_val)/len(df_clean)*100:.1f}%)")
print(f"Test: {len(X_test)} ({len(X_test)/len(df_clean)*100:.1f}%)")
print(f"Train+Val: {len(X_train_and_val)} ({len(X_train_and_val)/len(df_clean)*100:.1f}%)")
```

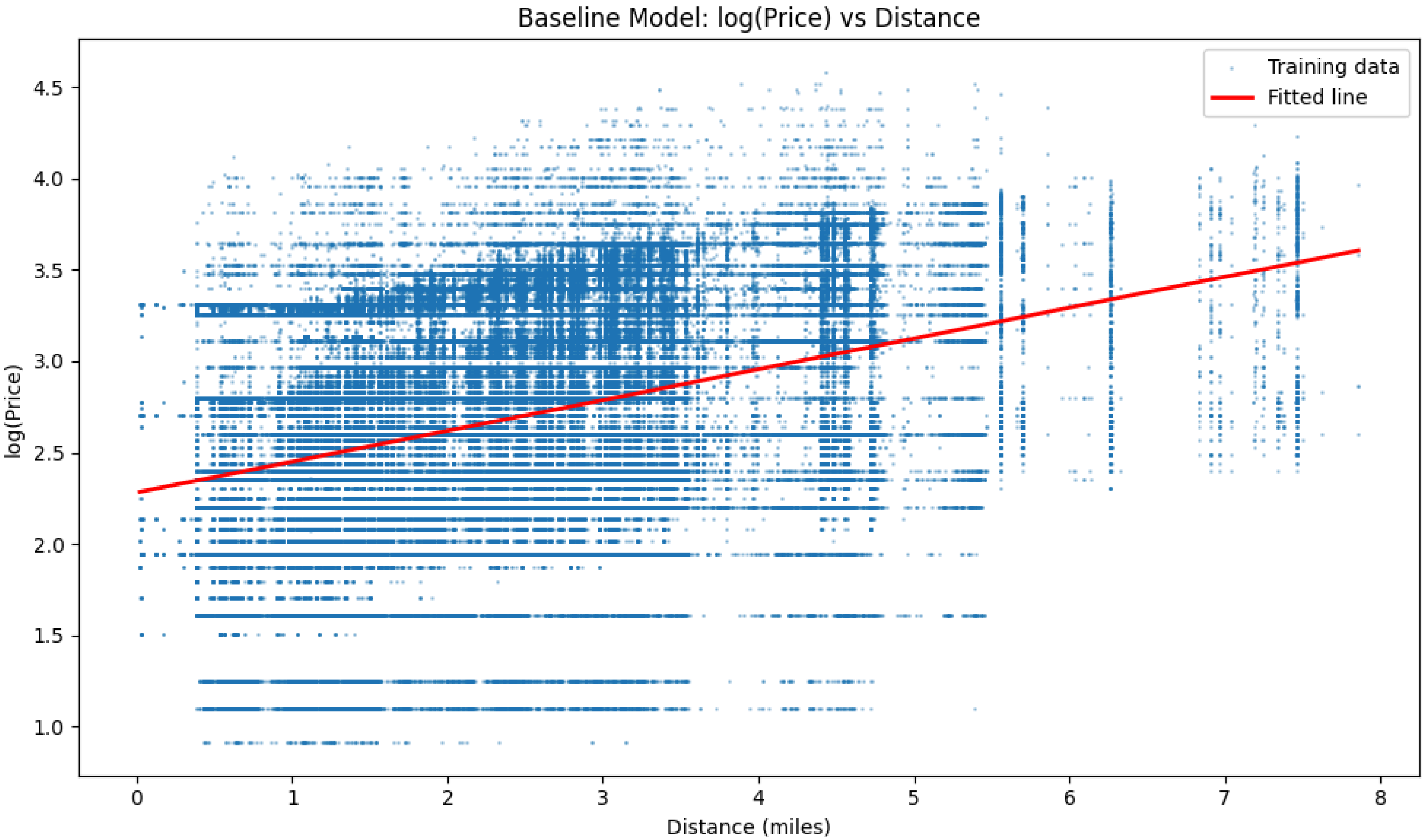
```
Data split sizes:
Train: 382785 (60.0%)
Validation: 127595 (20.0%)
Test: 127596 (20.0%)
Train+Val: 510380 (80.0%)
```

In [22]:

```
baseline_model = LinearRegression()
baseline_model.fit(X_train, y_train)

y_train_pred = baseline_model.predict(X_train)
y_val_pred = baseline_model.predict(X_val)

train_r2 = r2_score(y_train, y_train_pred)
val_r2 = r2_score(y_val, y_val_pred)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
val_rmse = np.sqrt(mean_squared_error(y_val, y_val_pred))

print("\nBaseline Model: log(price) ~ distance")
print(f"Coefficient (distance): {baseline_model.coef_[0]:.4f}")
print(f"Intercept: {baseline_model.intercept_:.4f}")
print(f"\nTrain R-sq: {train_r2:.4f}")
print(f"Validation R-sq: {val_r2:.4f}")
print(f"Train RMSE: {train_rmse:.4f}")
print(f"Validation RMSE: {val_rmse:.4f}")

# Visualize
plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, alpha=0.3, label='Training data', s=1)

X_train_sorted = np.sort(X_train, axis=0)
plt.plot(X_train_sorted,
         baseline_model.predict(X_train_sorted),
         color='red', linewidth=2, label='Fitted line')
```

```
plt.xlabel('Distance (miles)')
plt.ylabel('log(Price)')
plt.title('Baseline Model: log(Price) vs Distance')
plt.legend()
plt.tight_layout()
plt.show()
```

```
Baseline Model: log(price) ~ distance
Coefficient (distance): 0.1685
Intercept: 2.2819

Train R-sq: 0.1135
Validation R-sq: 0.1169
Train RMSE: 0.5347
Validation RMSE: 0.5368
```



The baseline model shows a positive linear relationship between distance and log-transformed price (coefficient = 0.169), but distance alone explains only about 11-12% of the variance in pricing (Train R^2 = 0.114, Validation R^2 = 0.117). The distinct horizontal banding in the scatter plot and low R^2 suggest that other factors—such as service type, surge pricing, and contextual variables—are necessary to adequately capture the ride-sharing pricing structure.

Multiple Linear Regression

Target encoding

We employ **target encoding** for the `source` and `destination` neighborhood variables rather than traditional one-hot encoding.

Our dataset contains 12 unique pickup neighborhoods (`source`) and 12 unique dropoff neighborhoods (`destination`). One-hot encoding would create 24 binary indicator variables. Target encoding reduces these 24 dimensions to just 2 continuous variables. It replaces each neighborhood with its average log(price) in the training set. This approach can efficiently encodes the economic characteristics of neighborhoods such as demand patterns, typical trip lengths, and surge likelihood into a single numerical value.

While we apply target encoding to locations, we deliberately use one-hot encoding for `name` (service class) and `cab_type` (platform). This hybrid approach allows us to create interpretable interaction terms such as `distance × UberXL` to capture how premium services charge different per-mile rates.

Target encoding represents an advanced feature engineering technique beyond the standard methods covered in our coursework. It demonstrates practical solutions to high-cardinality categorical variables, a common challenge in real-world machine learning applications.

In [23]:
```
df_clean = df.dropna()

feature_cols = [
    'distance', 'surge_multiplier', 'temperature', 'precipIntensity',
    'hour', 'is_weekend', 'month_name',
    'source', 'destination', 'name', 'cab_type'
]


# feature matrix and target variable (used for both MLR and Decision Tree)
X_features = df_clean[feature_cols].copy()
y_target = np.log(df_clean['price']).copy()

X_train_val, X_test, y_train_val, y_test = train_test_split(
    X_features, y_target, test_size=0.2, random_state=42
)
```

```
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.25, random_state=42
)

# reset indices
X_train = X_train.reset_index(drop=True)
X_val = X_val.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_val = y_val.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)

print(f"  Training set: {X_train.shape[0]:,} samples ({X_train.shape[0]/len(df_clean)*100:.1f}%)")
print(f"  Validation set: {X_val.shape[0]:,} samples ({X_val.shape[0]/len(df_clean)*100:.1f}%)")
print(f"  Test set: {X_test.shape[0]:,} samples ({X_test.shape[0]/len(df_clean)*100:.1f}%)")
```

```
 Training set: 382,785 samples (60.0%)
 Validation set: 127,595 samples (20.0%)
 Test set: 127,596 samples (20.0%)
```

In [24]:
```
# define columns for target encoding
location_cols = ['source', 'destination']
target_encoder = TargetEncoder(cols=location_cols, smoothing=1.0)

# fit on training data only
target_encoder.fit(X_train[location_cols], y_train)

# transform all sets
X_train_loc_encoded = target_encoder.transform(X_train[location_cols])
X_val_loc_encoded = target_encoder.transform(X_val[location_cols])
X_test_loc_encoded = target_encoder.transform(X_test[location_cols])

# rename columns
X_train_loc_encoded.columns = ['source_encoded', 'destination_encoded']
X_val_loc_encoded.columns = ['source_encoded', 'destination_encoded']
X_test_loc_encoded.columns = ['source_encoded', 'destination_encoded']
```

In [25]:
```
# mapping
# source neighborhoods
encoding_map_source = pd.DataFrame({
    'source': X_train['source'].values,
    'source_encoded': X_train_loc_encoded['source_encoded'].values
```

```
}).drop_duplicates().sort_values('source_encoded', ascending=False)

print("\nTop 5 most expensive pickup neighborhoods:")
print(encoding_map_source.head())
print("\nTop 5 least expensive pickup neighborhoods:")
print(encoding_map_source.tail())

# destination neighborhoods
encoding_map_dest = pd.DataFrame({
    'destination': X_train['destination'].values,
    'destination_encoded': X_train_loc_encoded['destination_encoded'].values
}).drop_duplicates().sort_values('destination_encoded', ascending=False)

print("\nTop 5 most expensive dropoff neighborhoods:")
print(encoding_map_dest.head())
print("\nTop 5 least expensive dropoff neighborhoods:")
print(encoding_map_dest.tail())
```

```
Top 5 most expensive pickup neighborhoods:
                     source  source_encoded
2         Boston University        2.781737
19                   Fenway        2.762028
9   Northeastern University        2.734369
7        Financial District        2.703400
1          Theatre District        2.659597

Top 5 least expensive pickup neighborhoods:
              source  source_encoded
10          Back Bay        2.624302
13       Beacon Hill        2.622324
4      South Station        2.607104
33         North End        2.575637
25  Haymarket Square        2.457238

Top 5 most expensive dropoff neighborhoods:
                destination  destination_encoded
8         Boston University             2.793605
0                    Fenway             2.749527
1   Northeastern University             2.737349
4        Financial District             2.701635
25            North Station             2.657249

Top 5 least expensive dropoff neighborhoods:
         destination  destination_encoded
5           West End             2.634631
2   Theatre District             2.631630
13         North End             2.554999
11     South Station             2.547252
3   Haymarket Square             2.507349
```

One-hot encoding

In [26]:
```
categorical_cols = ['name', 'cab_type', 'month_name']
# create dummy variables
X_train_categorical = pd.get_dummies(X_train[categorical_cols], drop_first=True, dtype=int)
X_val_categorical = pd.get_dummies(X_val[categorical_cols], drop_first=True, dtype=int)
X_test_categorical = pd.get_dummies(X_test[categorical_cols], drop_first=True, dtype=int)
```

```python
all_categorical_cols = X_train_categorical.columns

# add missing columns
for col in all_categorical_cols:
    if col not in X_val_categorical.columns:
        X_val_categorical[col] = 0
    if col not in X_test_categorical.columns:
        X_test_categorical[col] = 0

# reorder columns
X_val_categorical = X_val_categorical[all_categorical_cols]
X_test_categorical = X_test_categorical[all_categorical_cols]
```

Combine all features

Our multiple linear regression model includes features organized into categories that capture different aspects of ride-sharing pricing, selected based on theoretical expectations and insights from our EDA.

Core pricing drivers

`Distance` serves as our primary explanatory variable since trip length theoretically determines ride-sharing costs. We include `surge_multiplier` to capture dynamic pricing adjustments during high demand periods. The service class variable (`name`) is essential because our EDA revealed that premium services like Lyft Lux have substantially higher prices than standard options, indicating different base rates and per-mile charges. The `cab_type` variable (Uber vs. Lyft) allows us to test whether the two platforms have systematically different pricing structures.

Spatial and temporal context

Pickup and dropoff locations (`source` and `destination`) influence pricing through neighborhood-specific demand patterns and typical trip characteristics. We apply target encoding to efficiently capture these location effects while avoiding dimensionality issues from one-hot encoding 24 neighborhoods. For temporal patterns, we include `hour` to capture rush hour effects, `is_weekend` to distinguish weekday commuting from weekend leisure travel, and `month` to account for seasonal variation in demand and pricing.

Weather

We include `temperature` and `precipIntensity` to capture how adverse weather affects supply and demand. We explicitly drop `precipProbability` due to strong multicollinearity with `precipIntensity` (r = 0.84), and exclude `cloudCover` and `short_summary` because our EDA showed minimal impact on pricing.

Interaction terms

We create interaction terms between distance and service-related variables, particularly `distance × service class`, because our EDA revealed distinct "banding" patterns suggesting different service tiers charge different rates per mile. We also include `distance × cab_type` to capture how platform choice and surge pricing affect marginal costs.

In [27]:
```
numerical_cols = ['distance', 'surge_multiplier', 'temperature',
                  'precipIntensity', 'hour', 'is_weekend']

# combine everything
X_train_combined = pd.concat([
    X_train[numerical_cols].reset_index(drop=True),
    X_train_loc_encoded.reset_index(drop=True),
    X_train_categorical.reset_index(drop=True)
], axis=1)

X_val_combined = pd.concat([
    X_val[numerical_cols].reset_index(drop=True),
    X_val_loc_encoded.reset_index(drop=True),
    X_val_categorical.reset_index(drop=True)
], axis=1)

X_test_combined = pd.concat([
    X_test[numerical_cols].reset_index(drop=True),
    X_test_loc_encoded.reset_index(drop=True),
    X_test_categorical.reset_index(drop=True)
], axis=1)

print(f"  Total features: {X_train_combined.shape[1]}")
print(f"    - Numerical features: {len(numerical_cols)}")
print(f"    - Target-encoded locations: {X_train_loc_encoded.shape[1]}")
print(f"    - One-hot encoded categoricals: {X_train_categorical.shape[1]}")
```

```
Total features: 21
  - Numerical features: 6
  - Target-encoded locations: 2
  - One-hot encoded categoricals: 13
```

Interaction terms

In [28]:

```
X_train_final = X_train_combined.copy()
X_val_final = X_val_combined.copy()
X_test_final = X_test_combined.copy()

# distance × service name
interaction_count = 0
service_name_cols = [col for col in X_train_categorical.columns if 'name_' in col]

for col in service_name_cols:
    interaction_name = f'distance_x_{col}'
    X_train_final[interaction_name] = X_train_final['distance'] * X_train_final[col]
    X_val_final[interaction_name] = X_val_final['distance'] * X_val_final[col]
    X_test_final[interaction_name] = X_test_final['distance'] * X_test_final[col]
    interaction_count += 1

# distance × cab_type
cab_type_cols = [col for col in X_train_categorical.columns if 'cab_type_' in col]
for col in cab_type_cols:
    interaction_name = f'distance_x_{col}'
    X_train_final[interaction_name] = X_train_final['distance'] * X_train_final[col]
    X_val_final[interaction_name] = X_val_final['distance'] * X_val_final[col]
    X_test_final[interaction_name] = X_test_final['distance'] * X_test_final[col]
    interaction_count += 1

print(f"Created {interaction_count} interaction terms")
interaction_cols = [col for col in X_train_final.columns if '_x_' in col]
print(f"\nInteraction terms:")
for i, col in enumerate(interaction_cols, 1):
    print(f"  {i}. {col}")
```

```
Created 13 interaction terms

Interaction terms:
  1. distance_x_name_Black SUV
  2. distance_x_name_Lux
  3. distance_x_name_Lux Black
  4. distance_x_name_Lux Black XL
  5. distance_x_name_Lyft
  6. distance_x_name_Lyft XL
  7. distance_x_name_Shared
  8. distance_x_name_UberPool
  9. distance_x_name_UberX
  10. distance_x_name_UberXL
  11. distance_x_name_WAV
  12. distance_x_month_name_November
  13. distance_x_cab_type_Uber
```

Train multiple linear regression

In [29]:
```
mlr_model = LinearRegression()
mlr_model.fit(X_train_final, y_train)

print(f"Features: {X_train_final.shape[1]}")
print(f"Intercept: {mlr_model.intercept_:.4f}")

y_train_pred_mlr = mlr_model.predict(X_train_final)
y_val_pred_mlr = mlr_model.predict(X_val_final)
y_test_pred_mlr = mlr_model.predict(X_test_final)

def calc_metrics(y_true, y_pred, set_name):
    r2 = r2_score(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print(f"{set_name}:")
    print(f"  R²:   {r2:.4f}")
    print(f"  RMSE: {rmse:.4f}")
    return {'R²': r2, 'RMSE': rmse}

train_metrics = calc_metrics(y_train, y_train_pred_mlr, "Training")
val_metrics = calc_metrics(y_val, y_val_pred_mlr, "Validation")
```

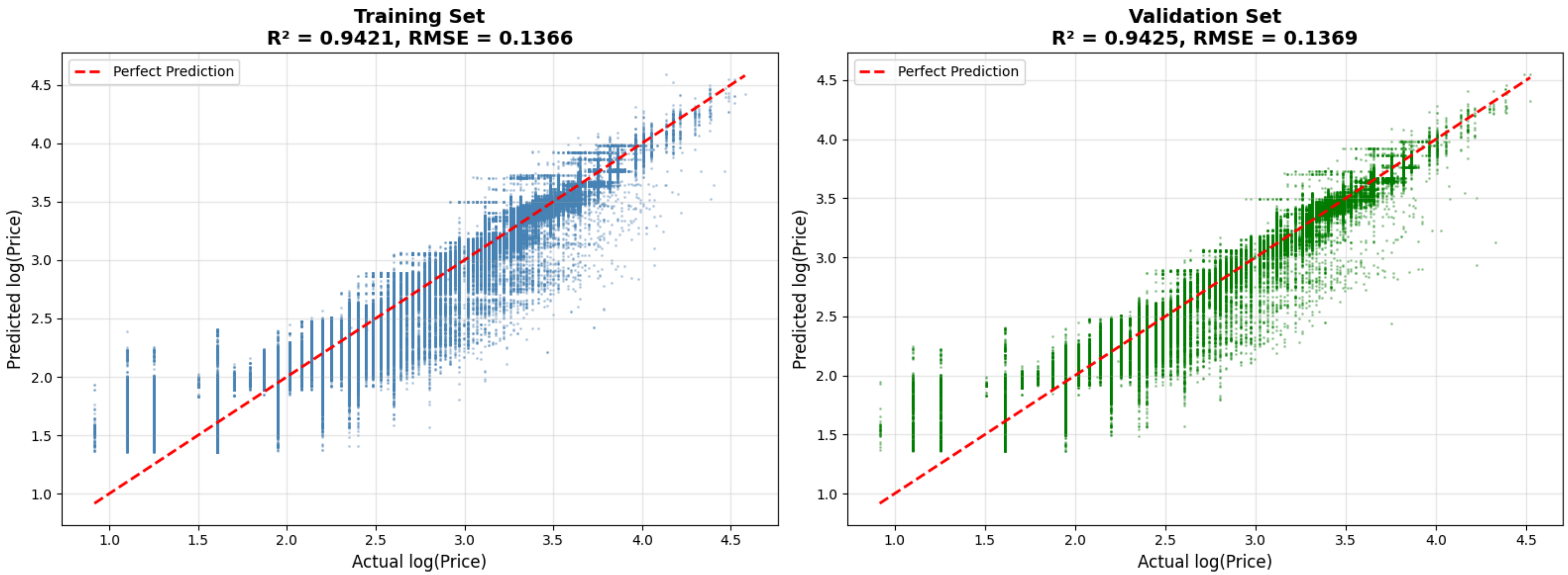
```
Features: 34
Intercept: 2.0154
Training:
  R²:   0.9421
  RMSE: 0.1366
Validation:
  R²:   0.9425
  RMSE: 0.1369
```

Multiple linear regression results

Our multiple linear regression model with 34 features achieves an R^2 of 0.9425 on the validation set, compared to just 0.1169 for the baseline distance-only model. This represents an 8-fold improvement in variance explained and a 74.5% reduction in RMSE (from 0.5368 to 0.1369). The near-identical performance on training (R^2 = 0.9421) and validation sets shows excellent generalization without overfitting. These results confirm that ride-sharing prices are determined not just by distance, but by a complex interplay of service class, platform, location, temporal patterns, and weather conditions.

In [30]:
```python
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Training set
axes[0].scatter(y_train, y_train_pred_mlr, alpha=0.3, s=1, color='steelblue')
axes[0].plot([y_train.min(), y_train.max()],
             [y_train.min(), y_train.max()],
             'r--', lw=2, label='Perfect Prediction')
axes[0].set_xlabel('Actual log(Price)', fontsize=12)
axes[0].set_ylabel('Predicted log(Price)', fontsize=12)
axes[0].set_title(f'Training Set\nR² = {train_metrics["R²"]:.4f}, RMSE = {train_metrics["RMSE"]:.4f}',
                  fontsize=14, fontweight='bold')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Validation set
axes[1].scatter(y_val, y_val_pred_mlr, alpha=0.3, s=1, color='green')
axes[1].plot([y_val.min(), y_val.max()],
             [y_val.min(), y_val.max()],
             'r--', lw=2, label='Perfect Prediction')
axes[1].set_xlabel('Actual log(Price)', fontsize=12)
axes[1].set_ylabel('Predicted log(Price)', fontsize=12)
axes[1].set_title(f'Validation Set\nR² = {val_metrics["R²"]:.4f}, RMSE = {val_metrics["RMSE"]:.4f}',
```

```
                    fontsize=14, fontweight='bold')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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

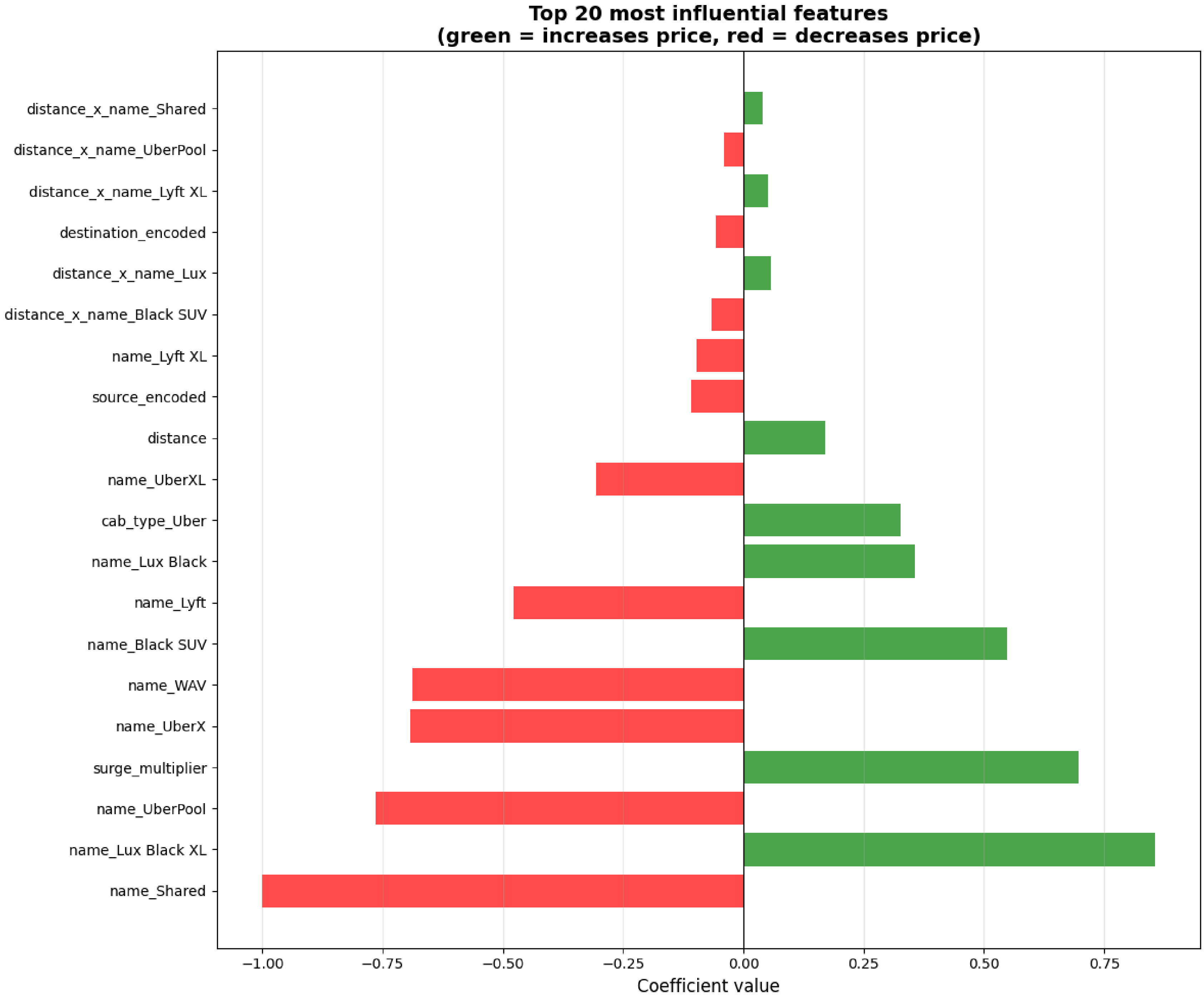


Feature importance

The coefficient analysis confirms that `service class` is the dominant pricing factor in the Boston ride-sharing market. Premium options like Lux Black XL (+0.85) and Black SUV (+0.52) command substantial premiums, while economy services like Shared rides (-1.05) and UberPool (-0.72) offer significant discounts. `Surge multiplier` (+0.70) emerges as the second most influential factor. `Distance` maintains a positive effect (+0.15), with interaction terms showing that premium services charge higher per-mile rates than economy options. The platform indicator shows `Uber` charges moderately more than `Lyft` (+0.33), while location effects (`source` and `destination` encodings) demonstrate neighborhood-level price variation.

In [31]:
```
# top 20 features by absolute coefficient value
coef_df = pd.DataFrame({
```

```
    'Feature': X_train_final.columns,
    'Coefficient': mlr_model.coef_
})
coef_df['Abs_Coefficient'] = coef_df['Coefficient'].abs()
top_20_features = coef_df.nlargest(20, 'Abs_Coefficient')

fig, ax = plt.subplots(figsize=(12, 10))
colors = ['green' if x > 0 else 'red' for x in top_20_features['Coefficient']]
bars = ax.barh(range(len(top_20_features)), top_20_features['Coefficient'], color=colors, alpha=0.7)
ax.set_yticks(range(len(top_20_features)))
ax.set_yticklabels(top_20_features['Feature'], fontsize=10)
ax.set_xlabel('Coefficient value', fontsize=12)
ax.set_title('Top 20 most influential features\n(green = increases price, red = decreases price)',
             fontsize=14, fontweight='bold')
ax.axvline(x=0, color='black', linestyle='-', linewidth=0.8)
ax.grid(True, alpha=0.3, axis='x')
plt.tight_layout()
plt.show()
```


Top 20 most influential features
(green = increases price, red = decreases price)
distance_x_name_Shared
distance_x_name_UberPool
distance_x_name_Lyft XL
destination_encoded
distance_x_name_Lux
distance_x_name_Black SUV
name_Lyft XL
source_encoded
distance
name_UberXL
cab_type_Uber
name_Lux Black
name_Lyft
name_Black SUV
name_WAV
name_UberX
surge_multiplier
name_UberPool
name_Lux Black XL
name_Shared
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
Coefficient value

Decision Tree Regression

While linear regression provides interpretable coefficients and strong performance, decision trees can capture non-linear relationships and complex interactions without explicitly defining them.

Thus, we'll build a decision tree model using cross-validation to optimize hyperparameters and prevent overfitting.

We use the **same data split** as MLR:

- Same target variable: `y_train` and `y_val`
- Same feature matrices: `X_train_final` and `X_val_final` (derived from the same initial split)
- Same random seed: `random_state=42` ensures reproducibility

In [32]:
```
warnings.filterwarnings('ignore')

# define hyperparameter grid for cross-validation
param_grid = {
    'max_depth': [5, 10, 15, 20, 25, 30],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 8],
    'max_features': ['sqrt', 'log2', None]
}

# initialize decision tree regressor
dt_base = DecisionTreeRegressor(random_state=42)

# perform 5-fold cross-validation
print("Performing 5-fold cross-validation to find optimal hyperparameters...")
print("This may take a few minutes...\n")

grid_search = GridSearchCV(
    estimator=dt_base,
    param_grid=param_grid,
    cv=5,
    scoring='neg_root_mean_squared_error',
    n_jobs=-1,
    verbose=1
```

```
)

# fit on training data
grid_search.fit(X_train_final, y_train)

print(f"\nBest hyperparameters found:")
for param, value in grid_search.best_params_.items():
    print(f"  {param}: {value}")
print(f"\nBest cross-validation RMSE: {-grid_search.best_score_:.4f}")
```

```
Performing 5-fold cross-validation to find optimal hyperparameters...
This may take a few minutes...

Fitting 5 folds for each of 288 candidates, totalling 1440 fits

Best hyperparameters found:
  max_depth: 15
  max_features: None
  min_samples_leaf: 8
  min_samples_split: 20

Best cross-validation RMSE: 0.1302
```

In [33]:

```
# train final model with best hyperparameters
dt_model = grid_search.best_estimator_

# make predictions
y_train_pred_dt = dt_model.predict(X_train_final)
y_val_pred_dt = dt_model.predict(X_val_final)
y_test_pred_dt = dt_model.predict(X_test_final)

# evaluate performance using same metrics as MLR
train_metrics_dt = calc_metrics(y_train, y_train_pred_dt, "Training")
val_metrics_dt = calc_metrics(y_val, y_val_pred_dt, "Validation")
test_metrics_dt = calc_metrics(y_test, y_test_pred_dt, "Test")
```

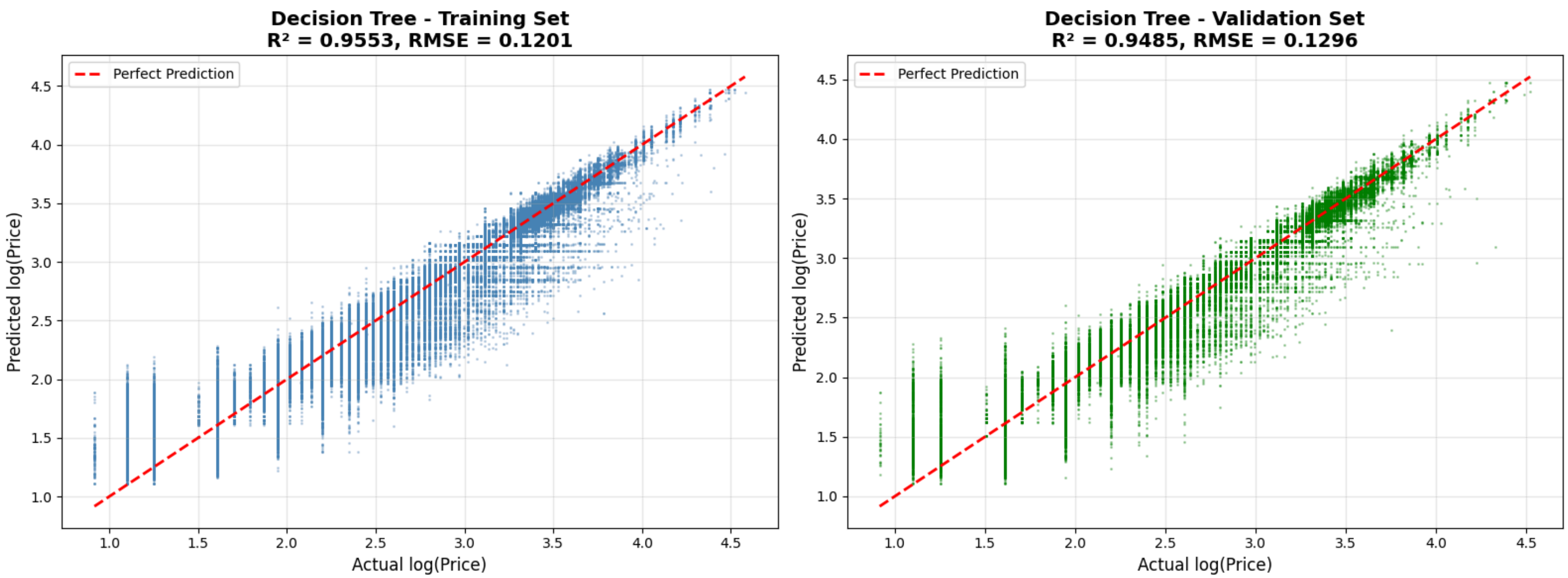
```
Training:
  R²:   0.9553
  RMSE: 0.1201
Validation:
  R²:   0.9485
  RMSE: 0.1296
Test:
  R²:   0.9486
  RMSE: 0.1289
```

Decision tree results

The optimized decision tree model shows strong predictive performance while maintaining good generalization. Cross-validation identified the optimal tree configuration to balance model complexity with prediction accuracy, preventing both underfitting and overfitting.

In [34]:
```
# visualization: predicted vs actual (same style as MLR)
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Training set
axes[0].scatter(y_train, y_train_pred_dt, alpha=0.3, s=1, color='steelblue')
axes[0].plot([y_train.min(), y_train.max()],
             [y_train.min(), y_train.max()],
             'r--', lw=2, label='Perfect Prediction')
axes[0].set_xlabel('Actual log(Price)', fontsize=12)
axes[0].set_ylabel('Predicted log(Price)', fontsize=12)
axes[0].set_title(f'Decision Tree - Training Set\nR² = {train_metrics_dt["R²"]:.4f}, RMSE = {train_metrics_
                  fontsize=14, fontweight='bold')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Validation set
axes[1].scatter(y_val, y_val_pred_dt, alpha=0.3, s=1, color='green')
axes[1].plot([y_val.min(), y_val.max()],
             [y_val.min(), y_val.max()],
             'r--', lw=2, label='Perfect Prediction')
axes[1].set_xlabel('Actual log(Price)', fontsize=12)
axes[1].set_ylabel('Predicted log(Price)', fontsize=12)
axes[1].set_title(f'Decision Tree - Validation Set\nR² = {val_metrics_dt["R²"]:.4f}, RMSE = {val_metrics_d
                  fontsize=14, fontweight='bold')
```

```
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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

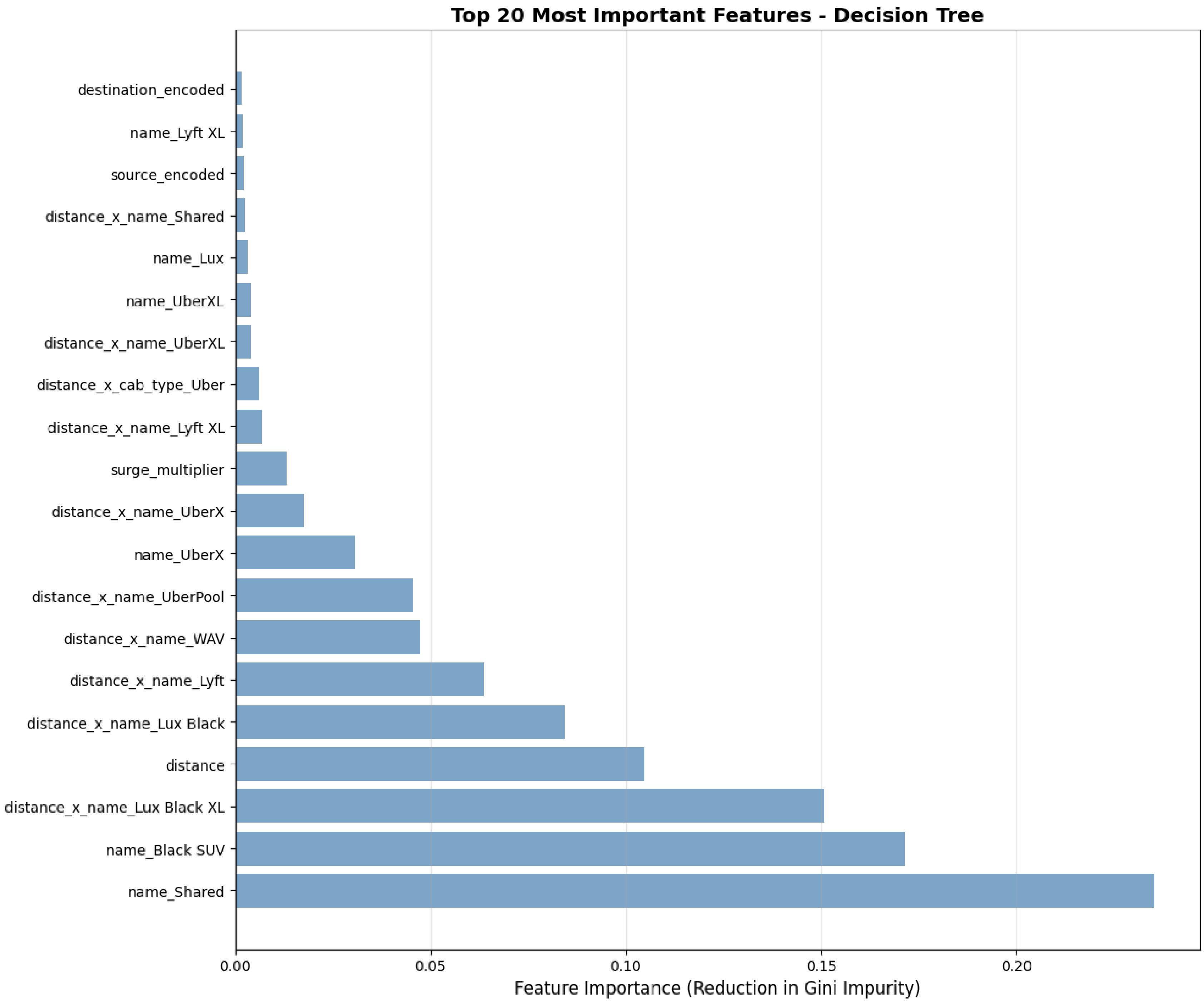


Feature importance

Decision trees provide feature importance measures based on the **mean decrease in Gini impurity**. Each feature's importance is calculated as the total reduction in node impurity (weighted by the probability of reaching that node) when that feature is used for splitting across all nodes in the tree. This directly reflects how much each feature contributes to reducing variance in the predictions.

In [35]:
```
# extract feature importances
feature_importance_df = pd.DataFrame({
    'Feature': X_train_final.columns,
    'Importance': dt_model.feature_importances_
})

# get top 20 features
```

```
top_20_dt_features = feature_importance_df.nlargest(20, 'Importance')

# plot feature importances
fig, ax = plt.subplots(figsize=(12, 10))
colors_dt = ['steelblue' for _ in range(len(top_20_dt_features))]
bars = ax.barh(range(len(top_20_dt_features)), top_20_dt_features['Importance'], color=colors_dt, alpha=0.
ax.set_yticks(range(len(top_20_dt_features)))
ax.set_yticklabels(top_20_dt_features['Feature'], fontsize=10)
ax.set_xlabel('Feature Importance (Reduction in Gini Impurity)', fontsize=12)
ax.set_title('Top 20 Most Important Features - Decision Tree',
             fontsize=14, fontweight='bold')
ax.grid(True, alpha=0.3, axis='x')
plt.tight_layout()
plt.show()

print("\nTop 20 Feature Importances:")
for i, (idx, row) in enumerate(top_20_dt_features.iterrows(), 1):
    print(f"{i:2d}. {row['Feature']:40s} {row['Importance']:.4f}")
```


Top 20 Most Important Features - Decision Tree
destination_encoded
name_Lyft XL
source_encoded
distance_x_name_Shared
name_Lux
name_UberXL
distance_x_name_UberXL
distance_x_cab_type_Uber
distance_x_name_Lyft XL
surge_multiplier
distance_x_name_UberX
name_UberX
distance_x_name_UberPool
distance_x_name_WAV
distance_x_name_Lyft
distance_x_name_Lux Black
distance
distance_x_name_Lux Black XL
name_Black SUV
name_Shared
0.00
0.05
0.10
0.15
0.20
Feature Importance (Reduction in Gini Impurity)

```
Top 20 Feature Importances:
 1. name_Shared                            0.2352
 2. name_Black SUV                         0.1715
 3. distance_x_name_Lux Black XL           0.1507
 4. distance                               0.1048
 5. distance_x_name_Lux Black              0.0843
 6. distance_x_name_Lyft                   0.0636
 7. distance_x_name_WAV                    0.0475
 8. distance_x_name_UberPool               0.0456
 9. name_UberX                             0.0306
10. distance_x_name_UberX                  0.0176
11. surge_multiplier                       0.0132
12. distance_x_name_Lyft XL                0.0068
13. distance_x_cab_type_Uber               0.0062
14. distance_x_name_UberXL                 0.0041
15. name_UberXL                            0.0041
16. name_Lux                               0.0032
17. distance_x_name_Shared                 0.0025
18. source_encoded                         0.0022
19. name_Lyft XL                           0.0021
20. destination_encoded                    0.0016
```

Model comparison: Decision Tree vs Multiple Linear Regression

Comparing the decision tree with our multiple linear regression model reveals the trade-offs between model flexibility and interpretability.

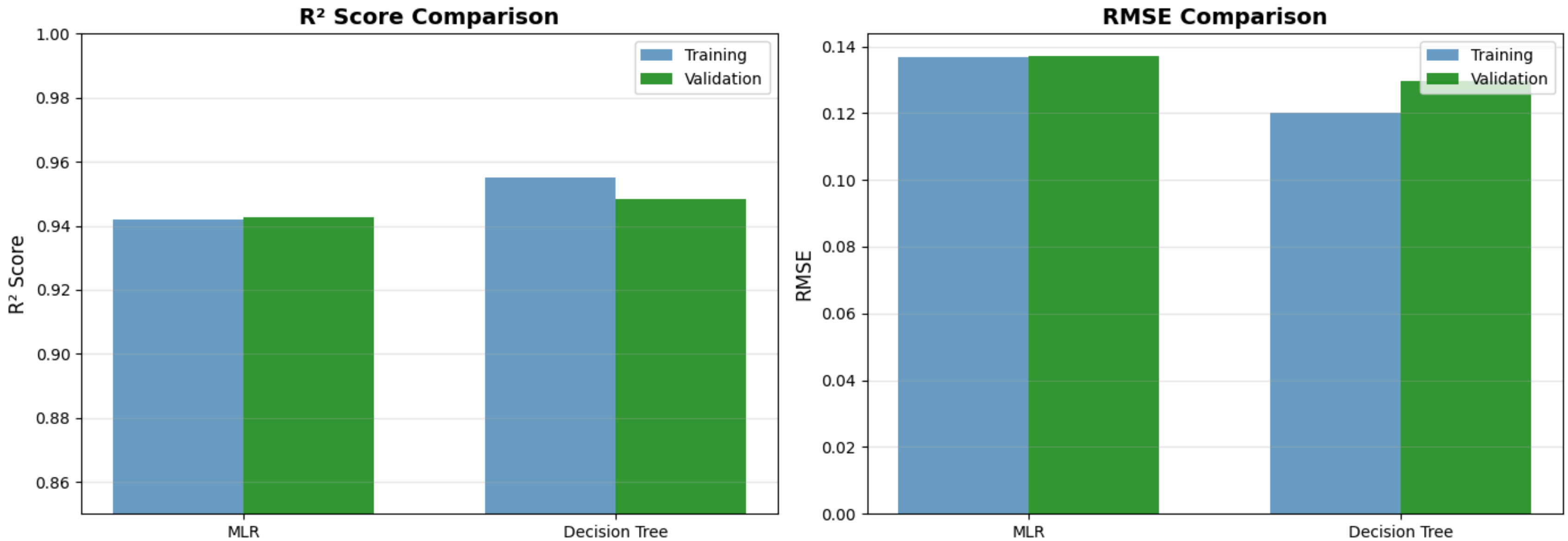
In [36]:

```
# create comparison table
comparison_df = pd.DataFrame({
    'Model': ['Multiple Linear Regression', 'Decision Tree'],
    'Train R²': [train_metrics['R²'], train_metrics_dt['R²']],
    'Train RMSE': [train_metrics['RMSE'], train_metrics_dt['RMSE']],
    'Validation R²': [val_metrics['R²'], val_metrics_dt['R²']],
    'Validation RMSE': [val_metrics['RMSE'], val_metrics_dt['RMSE']],
    'Test R²': [np.nan, test_metrics_dt['R²']],
    'Test RMSE': [np.nan, test_metrics_dt['RMSE']]
})

print("Model Performance Comparison:")
print("="*90)
```

```
print(comparison_df.to_string(index=False))
print("="*90)

# calculate overfitting gap
mlr_gap = train_metrics['R²'] - val_metrics['R²']
dt_gap = train_metrics_dt['R²'] - val_metrics_dt['R²']

print(f"\nOverfitting Analysis (Train R² - Validation R²):")
print(f"  MLR:           {mlr_gap:.4f}")
print(f"  Decision Tree: {dt_gap:.4f}")

# visualize comparison
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# R² comparison
models = ['MLR', 'Decision Tree']
train_r2 = [train_metrics['R²'], train_metrics_dt['R²']]
val_r2 = [val_metrics['R²'], val_metrics_dt['R²']]

x = np.arange(len(models))
width = 0.35

axes[0].bar(x - width/2, train_r2, width, label='Training', color='steelblue', alpha=0.8)
axes[0].bar(x + width/2, val_r2, width, label='Validation', color='green', alpha=0.8)
axes[0].set_ylabel('R² Score', fontsize=12)
axes[0].set_title('R² Score Comparison', fontsize=14, fontweight='bold')
axes[0].set_xticks(x)
axes[0].set_xticklabels(models)
axes[0].legend()
axes[0].grid(True, alpha=0.3, axis='y')
axes[0].set_ylim([0.85, 1.0])

# RMSE comparison
train_rmse = [train_metrics['RMSE'], train_metrics_dt['RMSE']]
val_rmse = [val_metrics['RMSE'], val_metrics_dt['RMSE']]

axes[1].bar(x - width/2, train_rmse, width, label='Training', color='steelblue', alpha=0.8)
axes[1].bar(x + width/2, val_rmse, width, label='Validation', color='green', alpha=0.8)
axes[1].set_ylabel('RMSE', fontsize=12)
axes[1].set_title('RMSE Comparison', fontsize=14, fontweight='bold')
axes[1].set_xticks(x)
axes[1].set_xticklabels(models)
```

```
axes[1].legend()
axes[1].grid(True, alpha=0.3, axis='y')

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

```
Model Performance Comparison:
==========================================================================================
                     Model  Train R²  Train RMSE  Validation R²  Validation RMSE  Test R²  Test RMSE
Multiple Linear Regression  0.942102    0.136648       0.942526         0.136949      NaN        NaN
             Decision Tree  0.955254    0.120130       0.948510         0.129624  0.94863   0.128937
==========================================================================================

Overfitting Analysis (Train R² - Validation R²):
  MLR:           -0.0004
  Decision Tree: 0.0067
```



Summary: MLR vs DECISION TREE

The decision tree model slightly outperforms multiple linear regression, achieving a validation R^2 of 0.9485 compared to MLR's 0.9425, with lower RMSE (0.1296 vs. 0.1369). Both models show excellent generalization with minimal overfitting—the decision tree has a train-validation R^2 gap of only 0.0067, while MLR actually generalizes slightly better to validation data (gap of -0.0004).

The key advantage of the decision tree is its ability to automatically capture non-linear relationships and complex interactions without manual feature engineering. The feature importance analysis reveals that service class indicators (name_Shared, name_Black SUV) and distance-based interactions dominate predictions, consistent with MLR's coefficient analysis. However, the decision tree identifies these patterns through recursive partitioning rather than linear coefficients.

While both models achieve strong predictive performance ($R^2 > 0.94$), MLR offers superior interpretability through its coefficients, making it easier to quantify the marginal effect of each feature. The decision tree provides more flexibility but at the cost of interpretability. For this pricing prediction task, either model would be suitable, with the choice depending on whether interpretability (favor MLR) or capturing complex non-linearities (favor decision tree) is prioritized.

Polynomial Regression

Build Polynomial Regression (Distance-Only Model)

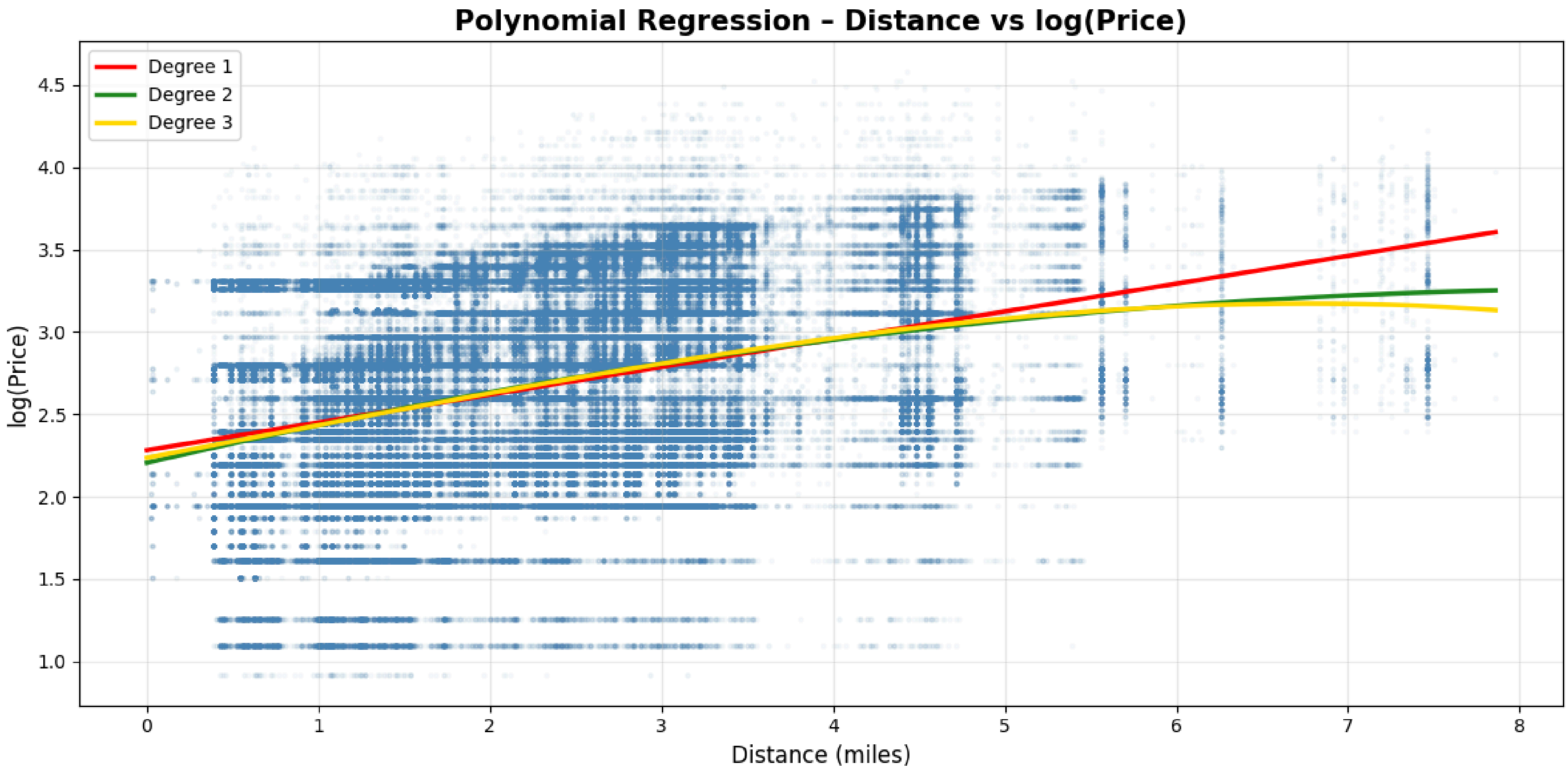
Since the distance-price relationship is non-linear, we want to improve over the distance-only baseline by adding polynomial terms. Extending the baseline model, we will start with degree 2-3 (most common), evaluate each and compare.

In [37]:
```
# Distance-only features for polynomial regression
X_train_dist = X_train[['distance']].copy()
X_val_dist = X_val[['distance']].copy()
X_test_dist = X_test[['distance']].copy()


def evaluate_poly_model(degree):
    # create polynomial transformer
    poly = PolynomialFeatures(degree=degree, include_bias=False)

    # transform distance-only input
    X_train_poly = poly.fit_transform(X_train_dist)
    X_val_poly = poly.transform(X_val_dist)

    # fit model
```

```
    model = LinearRegression()
    model.fit(X_train_poly, y_train)

    # predictions
    train_pred = model.predict(X_train_poly)
    val_pred = model.predict(X_val_poly)

    # metrics
    r2_train = r2_score(y_train, train_pred)
    r2_val = r2_score(y_val, val_pred)

    rmse_train = np.sqrt(mean_squared_error(y_train, train_pred))
    rmse_val = np.sqrt(mean_squared_error(y_val, val_pred))

    print(f"\nPolynomial Degree {degree}")
    print(f"  Train R²: {r2_train:.4f}, RMSE: {rmse_train:.4f}")
    print(f"  Val   R²: {r2_val:.4f}, RMSE: {rmse_val:.4f}")

    return {
        'degree': degree,
        'train_r2': r2_train,
        'val_r2': r2_val,
        'train_rmse': rmse_train,
        'val_rmse': rmse_val
    }

results_poly = []

for d in [1, 2, 3]:
    results_poly.append(evaluate_poly_model(d))
```

```
Polynomial Degree 1
  Train R²: 0.1135, RMSE: 0.5347
  Val   R²: 0.1169, RMSE: 0.5368

Polynomial Degree 2
  Train R²: 0.1160, RMSE: 0.5340
  Val   R²: 0.1194, RMSE: 0.5361

Polynomial Degree 3
  Train R²: 0.1161, RMSE: 0.5339
  Val   R²: 0.1196, RMSE: 0.5360
```

In [38]:
```
dist_plot = np.linspace(0, X_train['distance'].max(), 500).reshape(-1, 1)

plt.figure(figsize=(12, 6))

colors = {
    1: 'red',      # degree 1 (baseline linear)
    2: 'forestgreen',   # degree 2
    3: 'gold'           # degree 3
}

for degree in [1, 2, 3]:
    poly = PolynomialFeatures(degree=degree, include_bias=False)
    dist_poly = poly.fit_transform(dist_plot)

    model = LinearRegression()
    model.fit(poly.fit_transform(X_train_dist), y_train)

    plt.plot(
        dist_plot,
        model.predict(dist_poly),
        label=f'Degree {degree}',
        linewidth=2.5,
        color=colors[degree]
    )

plt.scatter(
    X_train['distance'], y_train,
    alpha=0.03, s=5,
    color='steelblue'
)

plt.xlabel("Distance (miles)", fontsize=12)
plt.ylabel("log(Price)", fontsize=12)
plt.title("Polynomial Regression – Distance vs log(Price)", fontsize=15, fontweight='bold')

plt.legend(frameon=True)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



Polynomial regression improves the distance-only model slightly, but the effect is marginal. The distance-price relationship is nearly linear, and higher-degree polynomial terms cannot compensate for the absence of key categorical and contextual features. This motivates the need for more complex models incorporating service type, surge, location, and time, which is exactly what we explore in the full MLR and decision tree models.

Polynomial Regression with Selected Features

The MLR results showed that service class, surge multiplier, platform (Uber vs. Lyft), and distance, particularly through its interactions with service tiers, are the primary drivers of log(price), while weather and temporal variables play secondary roles. Building on these findings, the polynomial regression incorporates this core subset of important features while introducing nonlinear transformations of distance (distance^2 and distance^3) to test whether allowing curvature in the distance–price relationship improves predictive performance beyond the linear specification.

We will use the features that previous MLR section identified as most important:

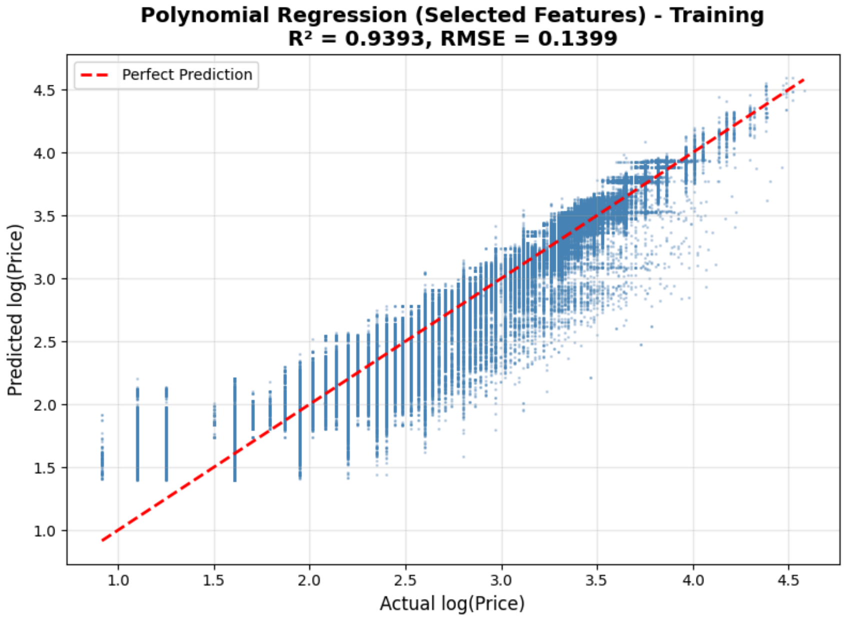
- `distance`
- `surge_multiplier`
- `temperature`
- `precipIntensity`
- `hour`
- `is_weekend`
- `cab_type`
- `name (service class)`
- `distance²`
- `distance³`

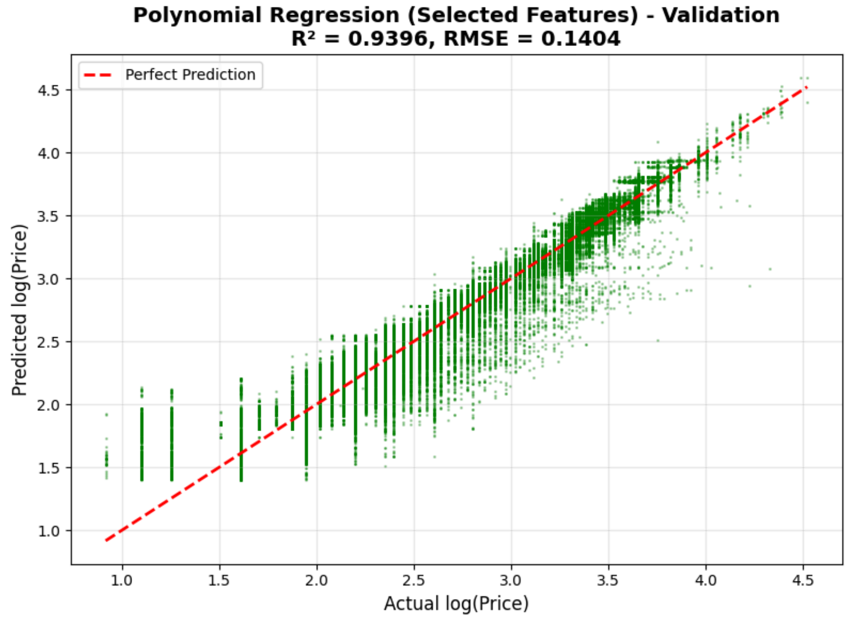
In [39]:
```
# Selected numerical features
numerical_cols = [
    'distance', 'surge_multiplier', 'temperature',
    'precipIntensity', 'hour', 'is_weekend'
]

# Selected categorical features
categorical_cols = ['cab_type', 'name']

# Extract base data
X_train_s2 = X_train[numerical_cols + categorical_cols].copy()
X_val_s2   = X_val[numerical_cols + categorical_cols].copy()
X_test_s2  = X_test[numerical_cols + categorical_cols].copy()

# Dummy encoding
X_train_s2 = pd.get_dummies(X_train_s2, columns=categorical_cols, drop_first=True)
X_val_s2   = pd.get_dummies(X_val_s2,   columns=categorical_cols, drop_first=True)
X_test_s2  = pd.get_dummies(X_test_s2,  columns=categorical_cols, drop_first=True)

# Ensure same columns across splits
X_val_s2  = X_val_s2.reindex(columns=X_train_s2.columns, fill_value=0)
X_test_s2 = X_test_s2.reindex(columns=X_train_s2.columns, fill_value=0)

for df in [X_train_s2, X_val_s2, X_test_s2]:
    df['distance_sq'] = df['distance'] ** 2
```

```
    df['distance_cu'] = df['distance'] ** 3

poly_s2_model = LinearRegression()
poly_s2_model.fit(X_train_s2, y_train)

# Predictions
y_train_pred_s2 = poly_s2_model.predict(X_train_s2)
y_val_pred_s2   = poly_s2_model.predict(X_val_s2)
y_test_pred_s2  = poly_s2_model.predict(X_test_s2)

def calc_metrics(y_true, y_pred, set_name=""):
    r2 = r2_score(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    print(f"{set_name}:")
    print(f"  R²:   {r2:.4f}")
    print(f"  RMSE: {rmse:.4f}")
    return {'R²': r2, 'RMSE': rmse}

# Evaluate
train_s2_metrics = calc_metrics(y_train, y_train_pred_s2, "Training (Poly S2)")
val_s2_metrics   = calc_metrics(y_val, y_val_pred_s2,   "Validation (Poly S2)")
test_s2_metrics  = calc_metrics(y_test, y_test_pred_s2, "Test (Poly S2)")

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Training set
axes[0].scatter(y_train, y_train_pred_s2, alpha=0.3, s=1, color='steelblue')
axes[0].plot([y_train.min(), y_train.max()],
             [y_train.min(), y_train.max()],
             'r--', lw=2, label='Perfect Prediction')
axes[0].set_xlabel('Actual log(Price)', fontsize=12)
axes[0].set_ylabel('Predicted log(Price)', fontsize=12)
axes[0].set_title(
    f'Polynomial Regression (Selected Features) - Training\n'
    f'R² = {train_s2_metrics["R²"]:.4f}, RMSE = {train_s2_metrics["RMSE"]:.4f}',
    fontsize=14, fontweight='bold'
)
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Validation set
axes[1].scatter(y_val, y_val_pred_s2, alpha=0.3, s=1, color='green')
```

```
axes[1].plot([y_val.min(), y_val.max()],
             [y_val.min(), y_val.max()],
             'r--', lw=2, label='Perfect Prediction')
axes[1].set_xlabel('Actual log(Price)', fontsize=12)
axes[1].set_ylabel('Predicted log(Price)', fontsize=12)
axes[1].set_title(
    f'Polynomial Regression (Selected Features) - Validation\n'
    f'R² = {val_s2_metrics["R²"]:.4f}, RMSE = {val_s2_metrics["RMSE"]:.4f}',
    fontsize=14, fontweight='bold'
)
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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

```
Training (Poly S2):
  R²:   0.9393
  RMSE: 0.1399
Validation (Poly S2):
  R²:   0.9396
  RMSE: 0.1404
Test (Poly S2):
  R²:   0.9398
  RMSE: 0.1396
```





These results represent a substantial improvement over the baseline distance-only models (R^2 is about 0.12) and show that incorporating service class, surge, platform, and temporal features dramatically increases explanatory power. Compared to the full MLR ($R^2 = 0.9425$) and the decision tree ($R^2 = 0.9485$), the polynomial model performs slightly worse, indicating that allowing distance to be nonlinear adds only limited predictive value once the key categorical and contextual variables are present.