In [532…
```
import os
import pandas as pd
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from scipy.stats import boxcox
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.inspection import permutation_importance
from sklearn.model_selection import GridSearchCV, cross_val_score
import statsmodels.api as sm
from tqdm import tqdm
```

Full data pre-processing and 'nearest_mrt_dist_m' and "distance_to_cbd_m" feature engineering has been completed in another notebook named "HTX_Full Coordinates and Distance Feature Engineering"

ML models

1. Model 1= Baseline linear regression for interpretability and for subsequent tuning comparisons
2. Model 2= Linear regression with minority classes '1bedroom' and 'executive' removed, 'lim chu kang' and 'bukit timah' removed, and high leverage points of 'floor_area_sqm' removed. As well as box-cox transformation for predictor 'resale_price'
3. Model 3= Non-Linear Decision Tree for feature importance and improved performance fit to non-linear hdb price data
4. Model 4= Non-linear GradientBoosting with better performance than single Decision Tree
5. Model 5= Linear regression with shorter timeframe selected (2018-2020) to investigate effects of "distance to MRT" and "distance to CBD" in the shorter time frame

In [12]:
```
geo_df_MRT_cbd = pd.read_csv('C:/Users/bulle/Desktop/HTX Interview/Kaggle_HDB/ge
```

```
C:\Users\bulle\AppData\Local\Temp\ipykernel_18112\74063598.py:1: DtypeWarning: Co
lumns (17) have mixed types. Specify dtype option on import or set low_memory=Fal
se.
  geo_df_MRT_cbd = pd.read_csv('C:/Users/bulle/Desktop/HTX Interview/Kaggle_HDB/g
eo_df_mrt_cbd.csv',index_col=0)
```

In [14]:
```
geo_df_MRT_cbd
```

Out[14]:

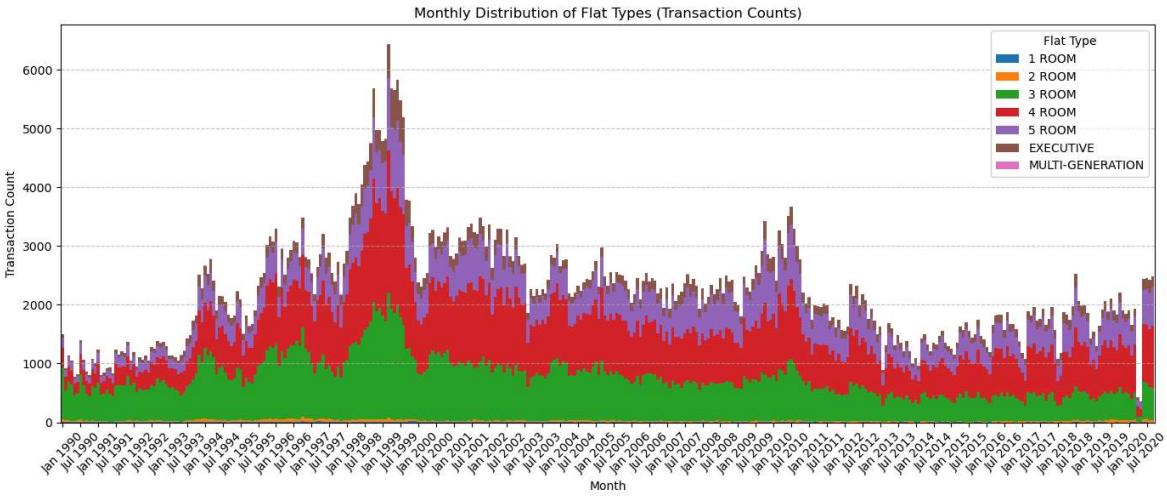
	month	town	flat_type	block	street_name	storey_range	floor_area_sq
0	1/1/1990	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	10 TO 12	31
1	1/1/1990	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	04 TO 06	31
2	1/1/1990	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	10 TO 12	31
3	1/1/1990	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	07 TO 09	31
4	1/1/1990	ANG MO KIO	3 ROOM	216	ANG MO KIO AVE 1	04 TO 06	73
...	...	...	...	...	...	...	
824727	1/12/2014	YISHUN	5 ROOM	816	YISHUN ST 81	10 TO 12	122
824728	1/12/2014	YISHUN	EXECUTIVE	325	YISHUN CTRL	10 TO 12	146
824729	1/12/2014	YISHUN	EXECUTIVE	618	YISHUN RING RD	07 TO 09	164
824730	1/12/2014	YISHUN	EXECUTIVE	277	YISHUN ST 22	07 TO 09	152
824731	1/12/2014	YISHUN	EXECUTIVE	277	YISHUN ST 22	04 TO 06	146

824732 rows × 20 columns

Step 2: EDA to understand the distribution and trends

In [16]:
```
#PLOT TRANSACTION_COUNT VS TIME

# Convert 'month' column to datetime
geo_df_MRT_cbd['month'] = geo_df_MRT_cbd['month'].astype(str).str.strip()
geo_df_MRT_cbd['month'] = pd.to_datetime(
    geo_df_MRT_cbd['month'],
    format='%d/%m/%Y',
    errors='coerce'
)

# Group by month and flat_type, count number of transactions
flat_type_counts = geo_df_MRT_cbd.groupby(['month', 'flat_type']).size().unstack
```

```
# Ensure the index is in datetime format
flat_type_counts.index = pd.to_datetime(flat_type_counts.index, format='%Y-%m')

# Plot again
fig, ax = plt.subplots(figsize=(14, 6))
flat_type_counts.plot(kind='bar', stacked=True, width=1.0, ax=ax)

# Format the x-axis with abbreviated month and year (e.g., Apr 2020)
ax.set_xticks(range(0, len(flat_type_counts.index), 6))  # One tick every 6 mont
formatted_labels = [d.strftime('%b %Y') for d in flat_type_counts.index[::6]]
ax.set_xticklabels(formatted_labels, rotation=45)

# Labels and legend
ax.set_title('Monthly Distribution of Flat Types (Transaction Counts)')
ax.set_xlabel('Month')
ax.set_ylabel('Transaction Count')
ax.legend(title='Flat Type', loc='upper right', frameon=True)
ax.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()

### From this plot, we can see the sharp spike in transaction volume in 1998-200
### It is worth correlating and quantifying the impact and power of cooling meas
```

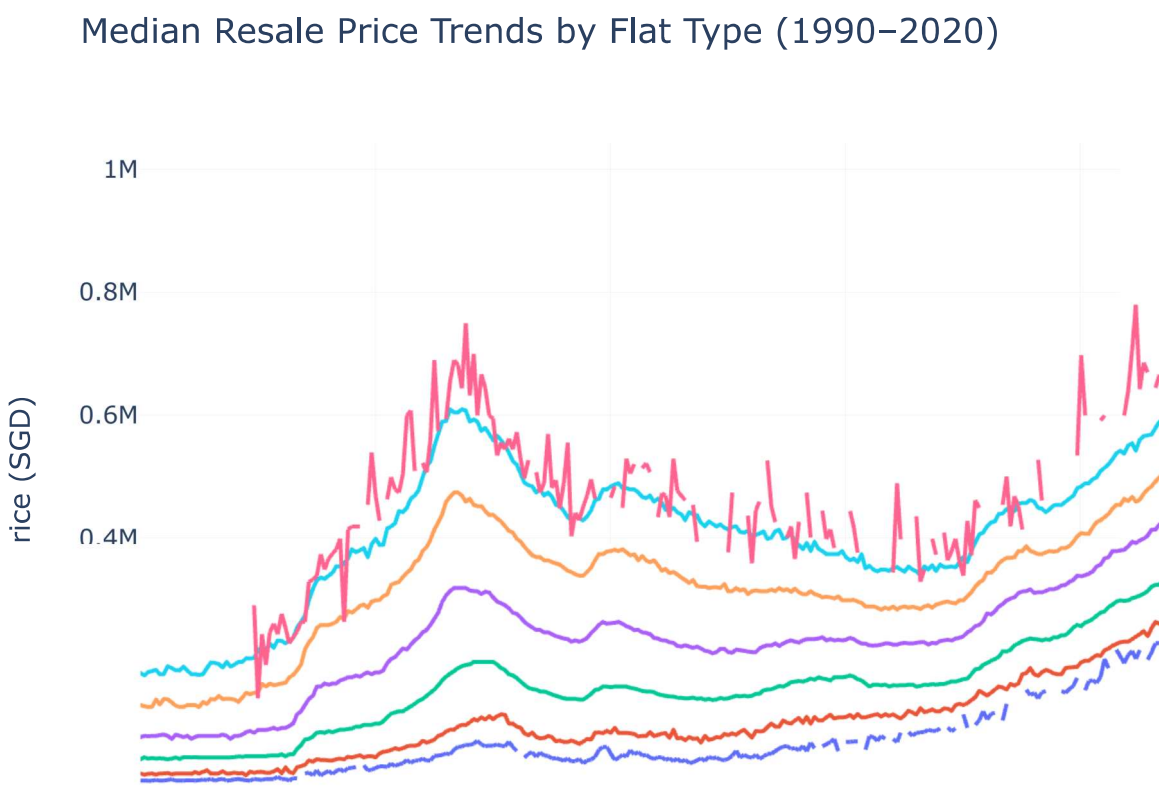


In [22]:

```
#PLOT RESALE PRICE VS TIME

# Group and aggregate resale price data
median_prices_line = geo_df_MRT_cbd.groupby(['month', 'flat_type'])['resale_pric
median_prices_line.index = pd.to_datetime(median_prices_line.index, errors='coer
median_prices_full = median_prices_line.loc["1990-01-01":"2020-09-01"]
median_melted_full = median_prices_full.reset_index().melt(
    id_vars='month', var_name='Flat Type', value_name='Median Resale Price'
)

fig = px.line(
    median_melted_full,
    x='month',
    y='Median Resale Price',
    color='Flat Type',
    title='Median Resale Price Trends by Flat Type (1990-2020)',
    labels={'month': 'Month', 'Median Resale Price': 'Price (SGD)'},
    template='plotly_white'
)
```

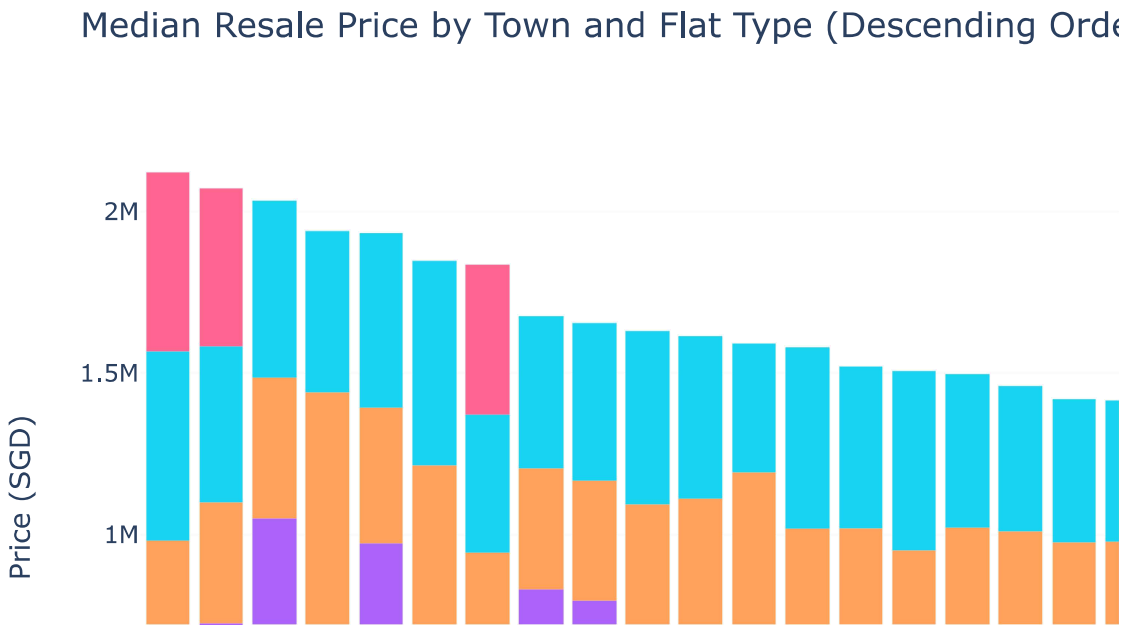
```
fig.update_layout(
    xaxis=dict(tickformat='%b-%Y', tickangle=45),
    legend_title_text='Flat Type'
)

fig.show()

### Median prices peaked in 1997, and fell/stagnated until recovery in 2013
```

```
C:\Users\bulle\anaconda3\Lib\site-packages\_plotly_utils\basevalidators.py:106: F
utureWarning: The behavior of DatetimeProperties.to_pydatetime is deprecated, in
a future version this will return a Series containing python datetime objects ins
tead of an ndarray. To retain the old behavior, call `np.array` on the result
  v = v.dt.to_pydatetime()
```



In [26]:
```
#PLOT RESALE PRICE VS TOWN (HUE= FLAT_TYPE)

# Assuming df_combined is your full dataset
town_flattype_median = geo_df_MRT_cbd.groupby(['town', 'flat_type'])['resale_pri

# Sort towns by total resale price (descending)
town_flattype_median['Total'] = town_flattype_median.sum(axis=1)
town_flattype_median_sorted = town_flattype_median.sort_values('Total', ascendin

# Melt for Plotly
town_melted = town_flattype_median_sorted.reset_index().melt(id_vars='town', var
```

```
# Plot stacked bar chart
fig = px.bar(
    town_melted,
    x='town',
    y='Median Price',
    color='Flat Type',
    title='Median Resale Price by Town and Flat Type (Descending Order)',
    labels={'Median Price': 'Price (SGD)'},
    template='plotly_white'
)

fig.update_layout(
    barmode='stack',
    xaxis_tickangle=45,
    height=600
)

fig.update_layout(barmode='stack', xaxis_tickangle=45)
fig.show()
```



🏠 Key Property Cooling Measures in Singapore (1996–2020)

1996–2000s: Early Measures 1996: Introduction of Seller's Stamp Duty (SSD) and restrictions on housing loans.

2010: Reintroduction of SSD for properties sold within a year; Loan-to-Value (LTV) limit reduced to 80%.

<The 5-year Minimum Occupation Period (MOP) became strictly enforced around 2010, particularly due to rising property speculation and "decoupling" to game the system>

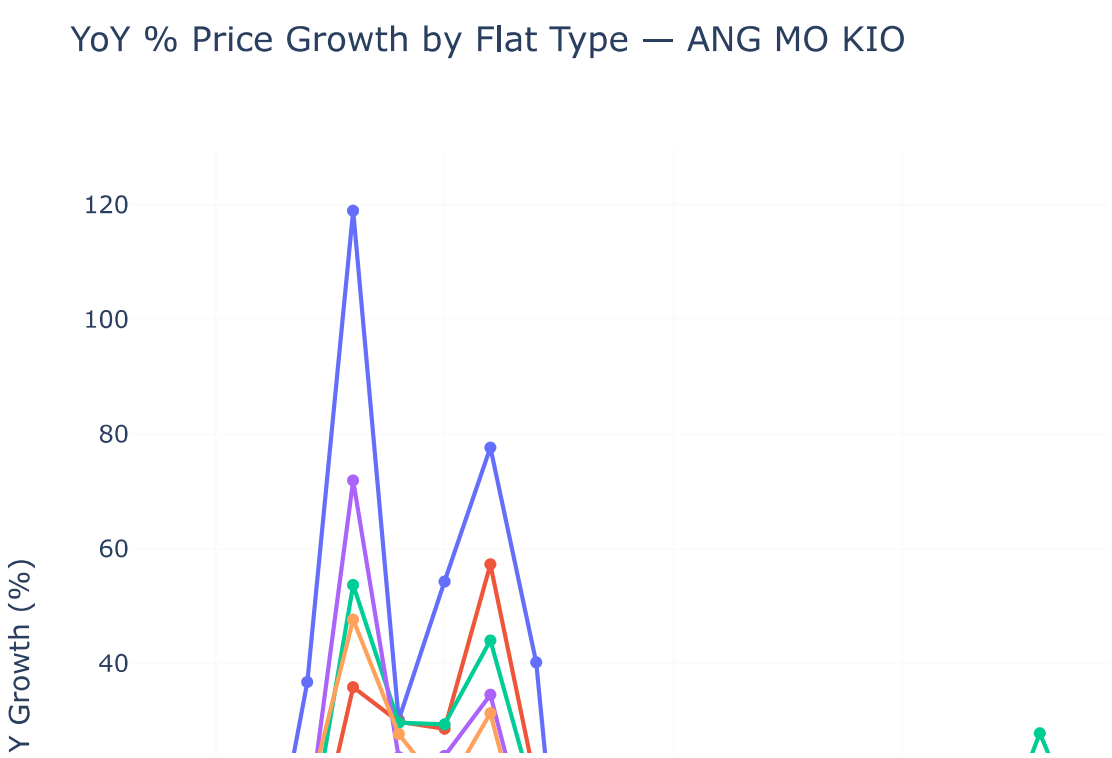
SSD holding period extended to 3 years; LTV lowered to 70% for buyers with existing loans.

2012: Introduction of Additional Buyer's Stamp Duty (ABSD): 10% for foreigners, 3% for PRs buying second property, and 3% for Singaporeans buying third property.

In [52]:
```
#PLOT YOY-GROWTH VS TIME
# Group by year and flat type
geo_df_MRT_cbd['year'] = pd.to_datetime(geo_df_MRT_cbd['month'], errors='coerce'
yearly_prices_ft = geo_df_MRT_cbd.groupby(['year', 'flat_type'])['resale_price']

# Calculate YoY %
yearly_prices_ft['YoY Growth (%)'] = yearly_prices_ft.groupby('flat_type')['resa
yearly_growth_ft = yearly_prices_ft.dropna(subset=['YoY Growth (%)'])

# Plot
fig = px.line(
    yearly_growth_ft,
    x='year',
    y='YoY Growth (%)',
    color='flat_type',
    title='Year-on-Year % Price Growth by Flat Type',
    template='plotly_white',
    markers=True
)

fig.update_layout(
    xaxis_title='Year',
    yaxis_title='YoY Growth (%)',
    legend_title_text='Flat Type',
    height=600
)

fig.show()
```

In [54]:
```
#PLOT YOY-GROWTH VS TIME (SELECTION BY TOWN)

# Group by town, flat_type, year and compute average resale price
grouped = geo_df_MRT_cbd.groupby(['town', 'flat_type', 'year'])['resale_price'].
grouped = grouped.sort_values(by=['town', 'flat_type', 'year'])
grouped['YoY Growth (%)'] = grouped.groupby(['town', 'flat_type'])['resale_price
yearly_growth = grouped.copy()

# Get list of towns and flat types
towns = sorted(yearly_growth['town'].unique())
flat_types = sorted(yearly_growth['flat_type'].unique())

# Create figure with all traces
fig = go.Figure()

# Add traces: one per town + flat_type
for town in towns:
    for ftype in flat_types:
        df = yearly_growth[(yearly_growth['town'] == town) & (yearly_growth['fla
        fig.add_trace(go.Scatter(
            x=df['year'],
            y=df['YoY Growth (%)'],
            mode='lines+markers',
```

```
            name=ftype,
            visible=(town == towns[0]),
            legendgroup=ftype,
            hovertemplate=f'Town: {town}<br>Flat Type: {ftype}<br>Year: %{{x}}<b
        ))

# dropdown buttons
buttons = []
for i, town in enumerate(towns):
    vis = [False] * (len(towns) * len(flat_types))
    start = i * len(flat_types)
    vis[start:start + len(flat_types)] = [True] * len(flat_types)

    buttons.append(dict(
        label=town,
        method='update',
        args=[{'visible': vis},
              {'title': f'YoY % Price Growth by Flat Type — {town}'}]
    ))

# Add layout and dropdown
fig.update_layout(
    updatemenus=[{
        'buttons': buttons,
        'direction': 'down',
        'showactive': True,
        'x': 0.95,
        'y': 0.9,
        'xanchor': 'right',
        'yanchor': 'bottom'
    }],
    title=f'YoY % Price Growth by Flat Type — {towns[0]}',
    xaxis_title='Year',
    yaxis_title='YoY Growth (%)',
    height=700,
    template='plotly_white'
)


fig.show()
```



Difference between correlation (what the model picks up) and causation (what actually drives systemic price control).

As a Data Scientist, my predictive model identifies key factors correlated with residential property prices in Singapore — such as location (e.g., proximity to MRT), flat type, town, floor area, and remaining lease. These features explain price variation at a transactional level.

However, it's important to note that while machine learning models are powerful at capturing relationships within available data, they may not account for exogenous interventions like government housing policies, which play a dominant role in curbing price inflation over time.

In particular, measures such as:

Additional Buyer's Stamp Duty (ABSD),

Loan-to-Value (LTV) reductions,

Total Debt Servicing Ratio (TDSR),

and HDB resale eligibility restrictions

— have had a causal impact on market cooling, by regulating demand and financing conditions. These policy levers are not always reflected directly in property-level datasets, but their macroeconomic effect is undeniable.

Therefore, my approach integrates:

Predictive modeling to uncover transaction-level price drivers,

Time series and intervention analysis to assess the impact of policy announcements on resale prices,

This hybrid approach supports both micro-level understanding and macro-level strategic planning — enabling evidence-based recommendations to curb housing price inflation.

In [62]:
```
geo_df_MRT_cbd = pd.read_csv('C:/Users/bulle/Desktop/HTX Interview/Kaggle_HDB/ge
```

```
C:\Users\bulle\AppData\Local\Temp\ipykernel_18112\74063598.py:1: DtypeWarning:

Columns (17) have mixed types. Specify dtype option on import or set low_memory=F
alse.
```

In [63]:
```
geo_df_MRT_cbd
```

Out[63]:

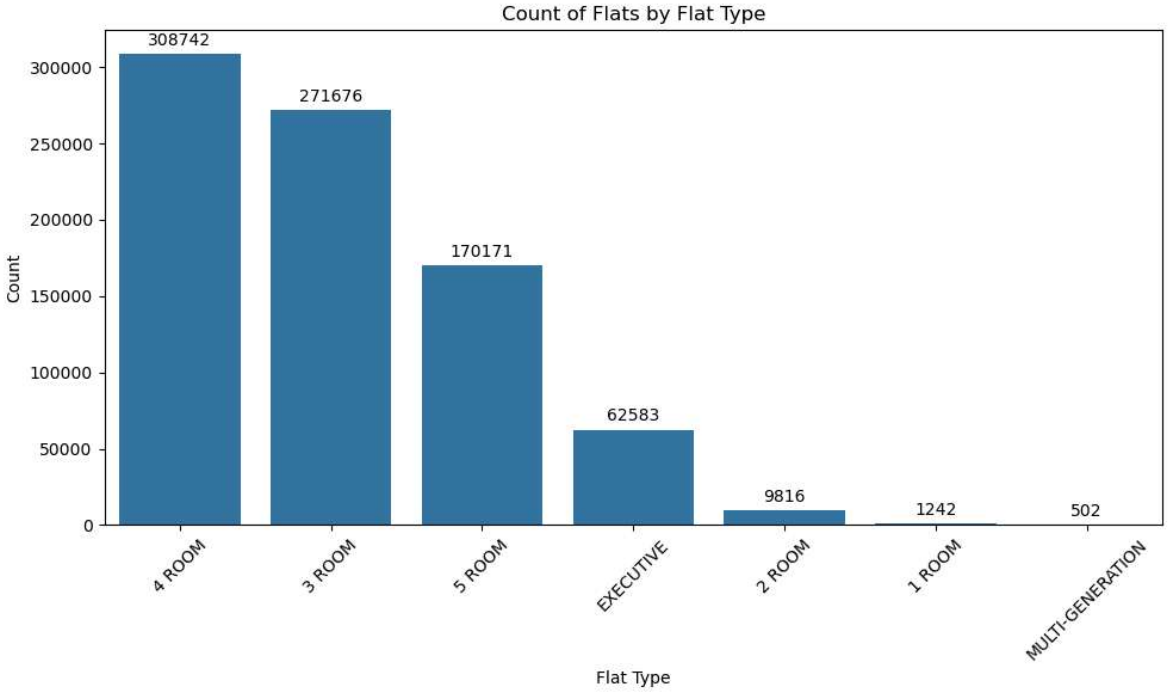
	month	town	flat_type	block	street_name	storey_range	floor_area_sq
0	1/1/1990	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	10 TO 12	31
1	1/1/1990	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	04 TO 06	31
2	1/1/1990	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	10 TO 12	31
3	1/1/1990	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	07 TO 09	31
4	1/1/1990	ANG MO KIO	3 ROOM	216	ANG MO KIO AVE 1	04 TO 06	73
...	...	...	...	...	...	...	
824727	1/12/2014	YISHUN	5 ROOM	816	YISHUN ST 81	10 TO 12	122
824728	1/12/2014	YISHUN	EXECUTIVE	325	YISHUN CTRL	10 TO 12	146
824729	1/12/2014	YISHUN	EXECUTIVE	618	YISHUN RING RD	07 TO 09	164
824730	1/12/2014	YISHUN	EXECUTIVE	277	YISHUN ST 22	07 TO 09	152
824731	1/12/2014	YISHUN	EXECUTIVE	277	YISHUN ST 22	04 TO 06	146

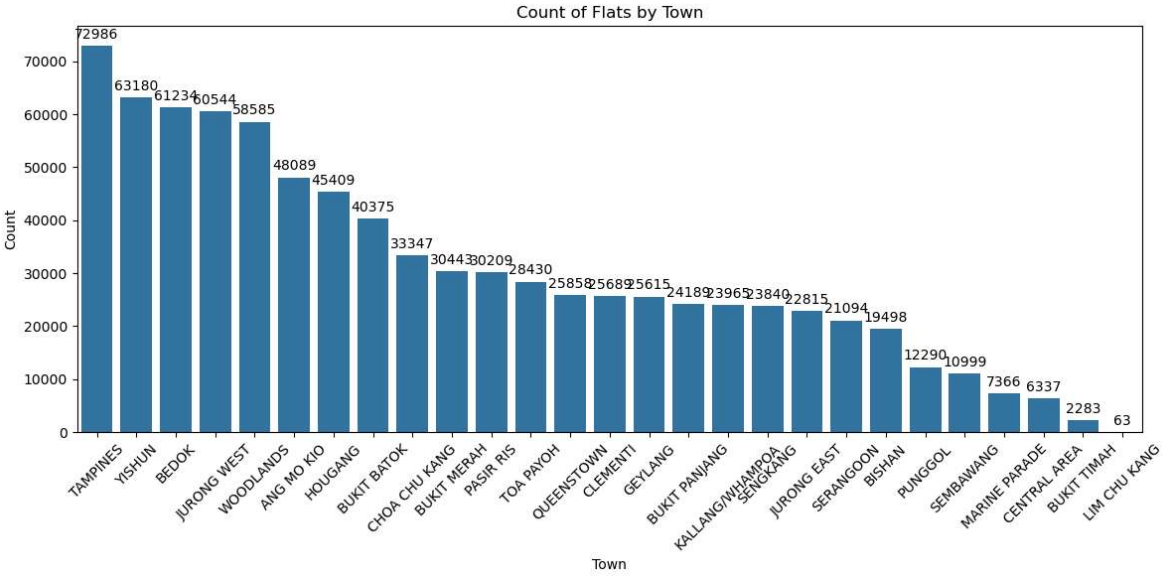
824732 rows × 20 columns

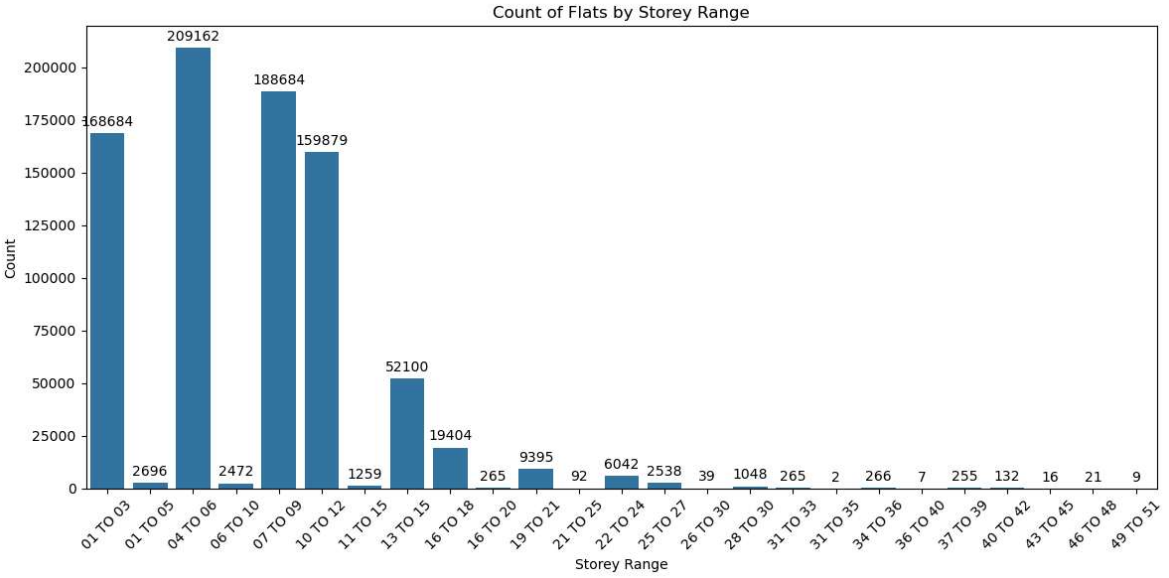
In [64]:

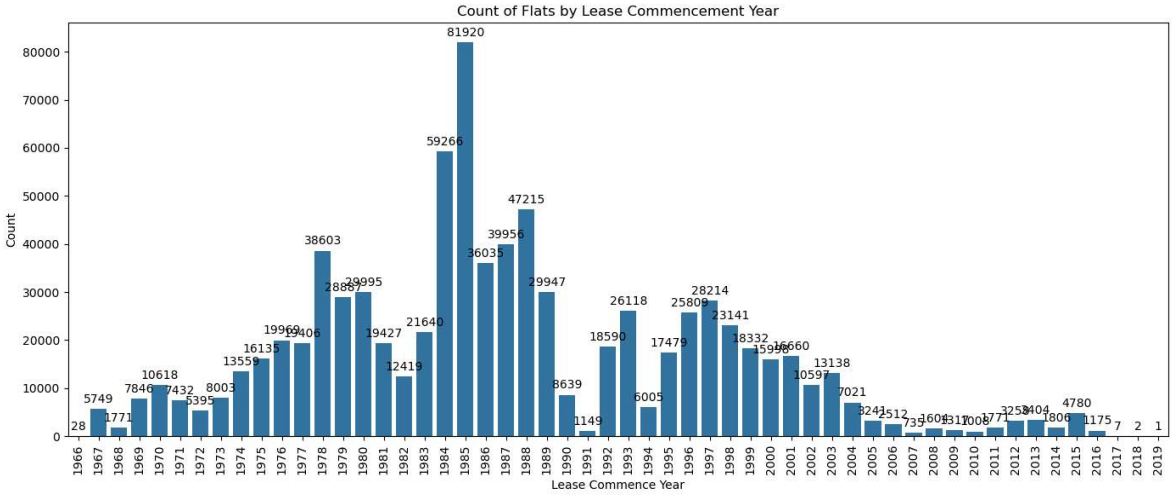
```
#explore the distribution count by flat type -> to see if we can omit 1 bed room

#first plot count by flat_type
plt.figure(figsize=(10, 6))
ax = sns.countplot(data=geo_df_MRT_cbd, x='flat_type', order=geo_df_MRT_cbd['fla
for container in ax.containers:
    ax.bar_label(container, fmt='%.0f', label_type='edge', padding=3)

plt.title("Count of Flats by Flat Type")
plt.xlabel("Flat Type")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
#second plot count by town
plt.figure(figsize=(12, 6))
ax = sns.countplot(
    data=geo_df_MRT_cbd,
    x='town',
    order=geo_df_MRT_cbd['town'].value_counts().index
)
for container in ax.containers:
    ax.bar_label(container, fmt='%.0f', label_type='edge', padding=3)

plt.title("Count of Flats by Town")
plt.xlabel("Town")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

#third plot count by storey_range
plt.figure(figsize=(12, 6))
ax = sns.countplot(
    data=geo_df_MRT_cbd,
    x='storey_range',
    order=geo_df_MRT_cbd['storey_range'].value_counts().sort_index().index  # sc
)

for container in ax.containers:
    ax.bar_label(container, fmt='%.0f', label_type='edge', padding=3)

plt.title("Count of Flats by Storey Range")
plt.xlabel("Storey Range")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# fourth plot count by lease_commence_date
plt.figure(figsize=(14, 6))
ax = sns.countplot(
    data=geo_df_MRT_cbd,
    x='lease_commence_date',
    order=geo_df_MRT_cbd['lease_commence_date'].value_counts().sort_index().inde
)

for container in ax.containers:
    ax.bar_label(container, fmt='%.0f', label_type='edge', padding=3)

plt.title("Count of Flats by Lease Commencement Year")
plt.xlabel("Lease Commence Year")
plt.ylabel("Count")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()


#there is clear class imbalance in our data for the discrete variables that can
#we can omit '1-room' and 'multi-generation' from our model.
#also noted that we have 3-room terraces up to 300sqm

#we will inspect non-linearity next
```


Count of Flats by Flat Type
Count
Flat Type
4 ROOM
3 ROOM
5 ROOM
EXECUTIVE
2 ROOM
1 ROOM
MULTI-GENERATION
308742
271676
170171
62583
9816
1242
502


Count of Flats by Town
Count
Town
TAMPINES
YISHUN
BEDOK
JURONG WEST
WOODLANDS
ANG MO KIO
HOUGANG
BUKIT BATOK
CHOA CHU KANG
BUKIT MERAH
PASIR RIS
TOA PAYOH
QUEENSTOWN
CLEMENTI
GEYLANG
BUKIT PANJANG
KALLANG/WHAMPOA
SENGKANG
JURONG EAST
SERANGOON
BISHAN
PUNGGOL
SEMBAWANG
MARINE PARADE
CENTRAL AREA
BUKIT TIMAH
LIM CHU KANG
72986
63180
61234
60544
58585
48089
45409
40375
33347
30443
30209
28430
25858
25689
25615
24189
23965
23840
22815
21094
19498
12290
10999
7366
6337
2283
63


Count of Flats by Storey Range
Count
Storey Range
01 TO 03
01 TO 05
04 TO 06
06 TO 10
07 TO 09
10 TO 12
11 TO 15
13 TO 15
16 TO 18
16 TO 20
19 TO 21
21 TO 25
22 TO 24
25 TO 27
26 TO 30
28 TO 30
31 TO 33
31 TO 35
34 TO 36
36 TO 40
37 TO 39
40 TO 42
43 TO 45
46 TO 48
49 TO 51
168684
2696
209162
2472
188684
159879
1259
52100
19404
265
9395
92
6042
2538
39
1048
265
2
266
7
255
132
16
21
9

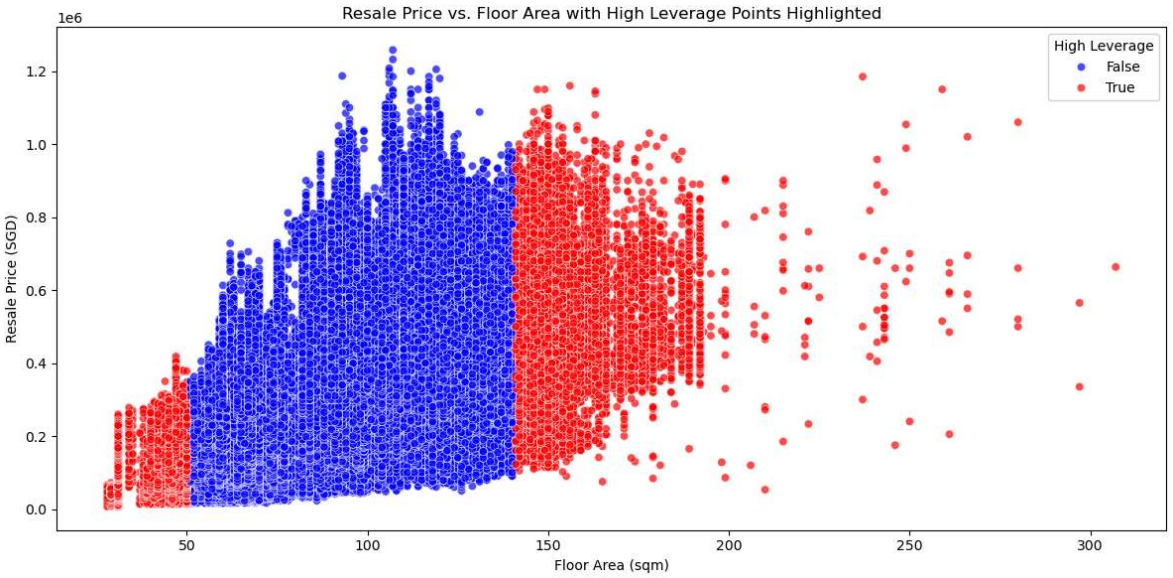


In [80]:
```
# Define variables
X = geo_df_MRT_cbd[['floor_area_sqm']]  # Only use this numerical predictor
y = geo_df_MRT_cbd['resale_price']

# Add constant
X_sm = sm.add_constant(X)

# Fit OLS model
model = sm.OLS(y, X_sm).fit()

# Influence diagnostics
influence = model.get_influence()
leverage = influence.hat_matrix_diag

# Identify high leverage points
n = X.shape[0]
p = X.shape[1]
leverage_threshold = 2 * (p + 1) / n #2 times average leverage threshold
high_leverage_points = np.where(leverage > leverage_threshold)[0]

print(f"Leverage threshold: {leverage_threshold:.6f}")
print(f"Number of high leverage points: {len(high_leverage_points)}")
```

```
Leverage threshold: 0.000005
Number of high leverage points: 67761
```

In [82]:
```
# Create a DataFrame for plotting leverage
plot_df = geo_df_MRT_cbd[['floor_area_sqm', 'resale_price']].copy()
plot_df['leverage'] = leverage
plot_df['high_leverage'] = plot_df.index.isin(high_leverage_points)

# Plot
plt.figure(figsize=(12, 6))
sns.scatterplot(
    data=plot_df,
    x='floor_area_sqm',
    y='resale_price',
    hue='high_leverage',
    palette={True: 'red', False: 'blue'},
    alpha=0.7
)
plt.title('Resale Price vs. Floor Area with High Leverage Points Highlighted')
plt.xlabel('Floor Area (sqm)')
plt.ylabel('Resale Price (SGD)')
```

```
plt.legend(title='High Leverage')
plt.tight_layout()
plt.show()

#we will remove these data points with high leverage in model 2 as they will dis
```



ML Model 1- Baseline Linear Regression for coefficient interpretation

In [248…
```
df_model1= geo_df_MRT_cbd.copy()
```

In [250…
```
#Further pre-processing to remove irrelevant feature columns and translate other

#get age of flat
current_year=2020
df_model1['age_of_flat']= current_year-df_model1['lease_commence_date']

#convert storey_range to numerical variable
def parse_storey_range(s):
    try:
        low, high = map(int, s.split(' TO '))
        return (low + high) / 2
    except:
        return None

df_model1['storey_avg'] = df_model1['storey_range'].apply(parse_storey_range)
df_model1.drop(columns=['storey_range'], axis=1, inplace=True)
df_model1.drop(columns=['block','street_name','flat_model','year','latitude','lo

#handle the 'month' variable to be continuous and numerical for LR model to unde
df_model1['month'] = pd.to_datetime(df_model1['month'])
df_model1['months_since_start'] = (
    df_model1['month'].dt.to_period('M') - df_model1['month'].min().to_period('M
).apply(lambda x: x.n)
df_model1.drop(columns=['month'], axis=1, inplace=True)

df_model1.reset_index(drop=True, inplace=True)
```

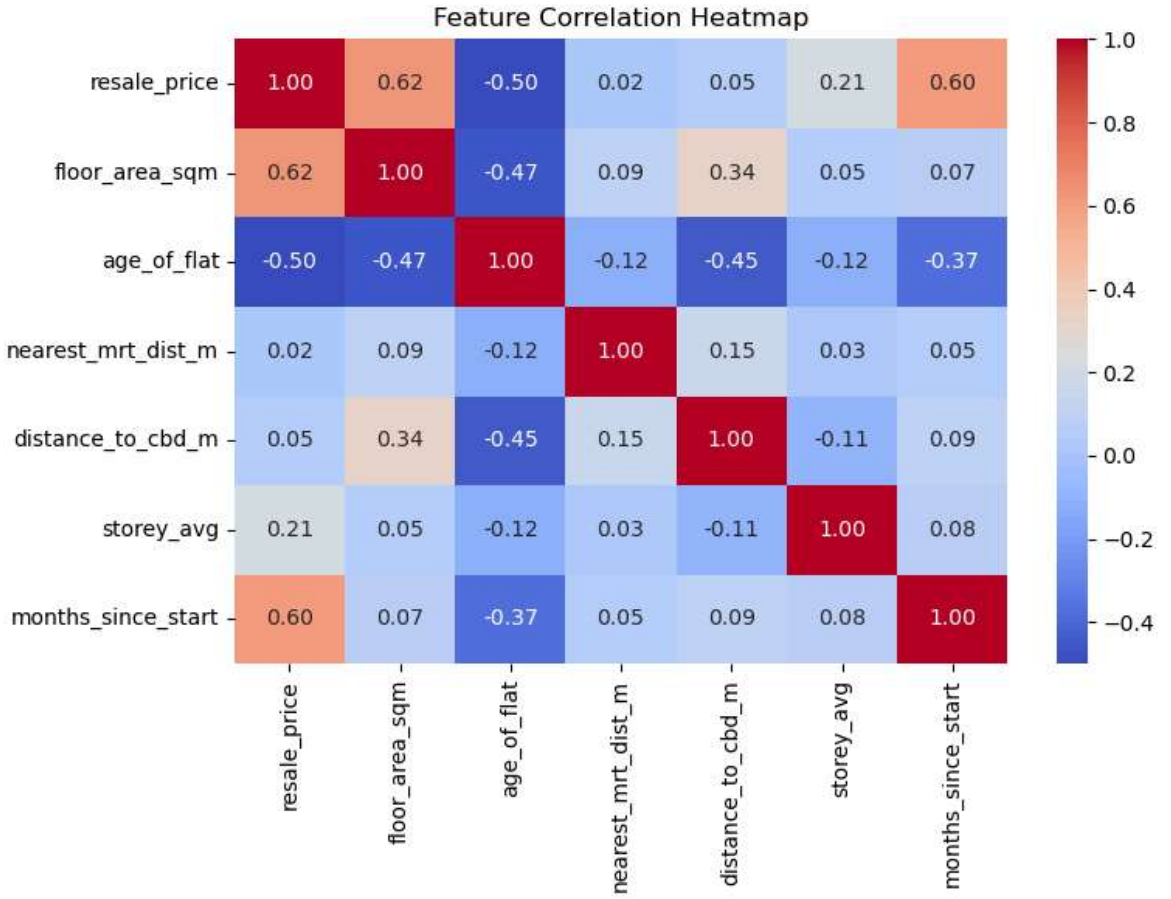
In [251…
```
df_model1
```

Out[251...

	town	floor_area_sqm	resale_price	nearest_mrt_dist_m	distance_to_cbd_m
0	ANG MO KIO	31.0	9000.0	826.278252	6834.444796
1	JURONG WEST	148.0	160000.0	1181.168902	13929.656560
2	JURONG WEST	151.0	150000.0	431.211489	13145.277170
3	JURONG WEST	151.0	159600.0	431.211489	13145.277170
4	JURONG WEST	149.0	193000.0	431.211489	13145.277170
...	...	...	...	...	...
824727	YISHUN	93.0	370000.0	1476.901513	12892.078360
824728	YISHUN	93.0	400888.0	1476.901513	12892.078360
824729	YISHUN	104.0	345000.0	1334.093222	13227.140710
824730	YISHUN	92.0	360000.0	1546.425808	12825.761210
824731	CHOA CHU KANG	142.0	470000.0	676.296577	13897.311300

824732 rows × 8 columns

In [88]:
```
#correlation heatmap to get a sensing of the strength of the features against re
num_features = ['resale_price', 'floor_area_sqm', 'age_of_flat',
                'nearest_mrt_dist_m', 'distance_to_cbd_m','storey_avg','months_s

corr_matrix = df_model1[num_features].corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.tight_layout()
plt.show()

#distance to MRT and CBD surprisingly has much weaker correlation with Resale Pr
```



In [89]:
```
#Standardisation of numerical features and one-hot encoding of categorical varia
df_model1.describe()
```

Out[89]:

	floor_area_sqm	resale_price	nearest_mrt_dist_m	distance_to_cbd_m	age_of_
count	824732.000000	8.247320e+05	824732.000000	824732.000000	824732.000
mean	95.584108	2.936435e+05	666.970303	10598.070249	32.847
std	26.054802	1.487287e+05	394.856600	3954.107276	9.448
min	28.000000	5.000000e+03	14.632192	1250.048224	1.000
25%	73.000000	1.810000e+05	390.853431	7552.357045	26.000
50%	93.000000	2.750000e+05	597.099992	11246.184360	34.000
75%	114.000000	3.830000e+05	857.839831	13918.841600	40.000
max	307.000000	1.258000e+06	5430.765648	19600.939980	54.000

In [90]:
```
df_model1
```

Out[90]:

	town	floor_area_sqm	resale_price	nearest_mrt_dist_m	distance_to_cbd_m	a
0	ANG MO KIO	31.0	9000.0	826.278252	6834.444796	
1	ANG MO KIO	31.0	6000.0	826.278252	6834.444796	
2	ANG MO KIO	31.0	8000.0	826.278252	6834.444796	
3	ANG MO KIO	31.0	6000.0	826.278252	6834.444796	
4	ANG MO KIO	73.0	47200.0	731.674686	6852.034457	
...	...	...	...	...	...	
824727	YISHUN	122.0	580000.0	586.741132	11818.482110	
824728	YISHUN	146.0	540000.0	725.118656	13791.219400	
824729	YISHUN	164.0	738000.0	291.324644	12593.162840	
824730	YISHUN	152.0	592000.0	924.858624	14717.777520	
824731	YISHUN	146.0	545000.0	924.858624	14717.777520	

824732 rows × 8 columns

In [91]:
```
# Step 1: Split before transformation
X = df_model1.drop('resale_price', axis=1)
y = df_model1['resale_price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Step 2: One-hot encode on training, then align columns in test set
X_train = pd.get_dummies(X_train, columns=['town'], drop_first=True)
X_test = pd.get_dummies(X_test, columns=['town'], drop_first=True)

# Align test columns with training (fill missing dummies with 0)
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)

# Step 3: Standardize only numerical features
num_features = ['floor_area_sqm', 'nearest_mrt_dist_m', 'distance_to_cbd_m',
                'age_of_flat', 'storey_avg', 'months_since_start']

scaler = StandardScaler()
X_train[num_features] = scaler.fit_transform(X_train[num_features])
X_test[num_features] = scaler.transform(X_test[num_features])
```

In [92]:
```
model1 = LinearRegression()
model1.fit(X_train, y_train)
```

Out[92]:

LinearRegression

LinearRegression()

In [93]:
```
X_train
```

Out[93]:

	floor_area_sqm	nearest_mrt_dist_m	distance_to_cbd_m	age_of_flat	storey_avg
677822	0.591302	-0.137170	0.335309	-1.464280	-0.549552
716000	-1.173890	0.630510	0.887616	1.074952	2.046994
638601	-0.406415	-1.081006	-0.702341	-1.993286	0.099585
665101	0.706423	2.499175	0.763163	-1.781684	2.046994
520734	-0.214547	-0.550243	0.212157	-1.464280	-0.549552
...	...	...	...	...	...
661055	-1.058769	0.085120	-1.203237	1.498157	0.748721
204614	0.284312	-0.252906	-1.174645	-0.512068	0.099585
476497	-0.444789	-0.660236	0.149363	0.016939	0.748721
214539	-1.173890	0.964319	-1.869211	1.286555	-0.549552
176991	-0.444789	-0.296914	-0.953973	0.228541	0.099585

577312 rows × 32 columns

In [94]:
```
coefficients = pd.DataFrame({
    'Feature': X_train.columns,
    'Coefficient': model1.coef_
}).sort_values(by='Coefficient', key=abs, ascending=False)

for feature, coef in zip(X_train.columns, model1.coef_):
    direction = "increases" if coef > 0 else "decreases"
    print(f"For every 1 unit increase in '{feature}', resale price {direction} b
```

```
For every 1 unit increase in 'floor_area_sqm', resale price increases by 89,087 S
GD, holding other variables constant.
For every 1 unit increase in 'nearest_mrt_dist_m', resale price decreases by 10,3
89 SGD, holding other variables constant.
For every 1 unit increase in 'distance_to_cbd_m', resale price decreases by 16,14
3 SGD, holding other variables constant.
For every 1 unit increase in 'age_of_flat', resale price decreases by 27,183 SGD,
holding other variables constant.
For every 1 unit increase in 'storey_avg', resale price increases by 14,138 SGD,
holding other variables constant.
For every 1 unit increase in 'months_since_start', resale price increases by 79,5
52 SGD, holding other variables constant.
For every 1 unit increase in 'town_BEDOK', resale price increases by 6,284 SGD, h
olding other variables constant.
For every 1 unit increase in 'town_BISHAN', resale price increases by 39,722 SGD,
holding other variables constant.
For every 1 unit increase in 'town_BUKIT BATOK', resale price decreases by 30,999
SGD, holding other variables constant.
For every 1 unit increase in 'town_BUKIT MERAH', resale price increases by 36,953
SGD, holding other variables constant.
For every 1 unit increase in 'town_BUKIT PANJANG', resale price decreases by 63,4
68 SGD, holding other variables constant.
For every 1 unit increase in 'town_BUKIT TIMAH', resale price increases by 60,560
SGD, holding other variables constant.
For every 1 unit increase in 'town_CENTRAL AREA', resale price increases by 34,89
0 SGD, holding other variables constant.
For every 1 unit increase in 'town_CHOA CHU KANG', resale price decreases by 65,7
10 SGD, holding other variables constant.
For every 1 unit increase in 'town_CLEMENTI', resale price increases by 18,955 SG
D, holding other variables constant.
For every 1 unit increase in 'town_GEYLANG', resale price increases by 5,227 SGD,
holding other variables constant.
For every 1 unit increase in 'town_HOUGANG', resale price decreases by 28,081 SG
D, holding other variables constant.
For every 1 unit increase in 'town_JURONG EAST', resale price decreases by 22,969
SGD, holding other variables constant.
For every 1 unit increase in 'town_JURONG WEST', resale price decreases by 38,336
SGD, holding other variables constant.
For every 1 unit increase in 'town_KALLANG/WHAMPOA', resale price increases by 7,
571 SGD, holding other variables constant.
For every 1 unit increase in 'town_LIM CHU KANG', resale price increases by 127,2
86 SGD, holding other variables constant.
For every 1 unit increase in 'town_MARINE PARADE', resale price increases by 122,
064 SGD, holding other variables constant.
For every 1 unit increase in 'town_PASIR RIS', resale price decreases by 210 SGD,
holding other variables constant.
For every 1 unit increase in 'town_PUNGGOL', resale price decreases by 28,006 SG
D, holding other variables constant.
For every 1 unit increase in 'town_QUEENSTOWN', resale price increases by 39,634
SGD, holding other variables constant.
For every 1 unit increase in 'town_SEMBAWANG', resale price decreases by 81,842 S
GD, holding other variables constant.
For every 1 unit increase in 'town_SENGKANG', resale price decreases by 51,653 SG
D, holding other variables constant.
For every 1 unit increase in 'town_SERANGOON', resale price increases by 266 SGD,
holding other variables constant.
For every 1 unit increase in 'town_TAMPINES', resale price increases by 1,796 SG
D, holding other variables constant.
For every 1 unit increase in 'town_TOA PAYOH', resale price increases by 16,426 S
GD, holding other variables constant.
```

```
For every 1 unit increase in 'town_WOODLANDS', resale price decreases by 70,124 S
GD, holding other variables constant.
For every 1 unit increase in 'town_YISHUN', resale price decreases by 23,140 SGD,
holding other variables constant.
```

In [502…

```python
# Create full coefficient DataFrame
coef_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Coefficient': model1.coef_,
    'Standardized Coef': model1.coef_
})

# Add std dev only for numerical
coef_df['Original Std'] = coef_df['Feature'].map(dict(zip(num_features, scaler.s

# Compute effect in original units (only for numerical features)
coef_df['Per Unit Effect (SGD)'] = coef_df.apply(
    lambda row: row['Coefficient'] / row['Original Std']
    if pd.notna(row['Original Std']) else row['Coefficient'],
    axis=1
)

# Add units for numeric features
units = {
    'floor_area_sqm': 'sqm',
    'age_of_flat': 'year of age',
    'storey_avg': 'storey',
    'nearest_mrt_dist_m': 'meter from MRT',
    'distance_to_cbd_m': 'meter from CBD',
    'months_since_start': 'month (of transaction)'
}
coef_df['Unit'] = coef_df['Feature'].map(units)

# --- Final interpretation printouts ---

#  🔹 Numerical features
print("📈 Effect of Numerical Features:\n")
for _, row in coef_df[coef_df['Feature'].isin(num_features)].iterrows():
    effect = row['Per Unit Effect (SGD)']
    direction = "increases" if effect > 0 else "decreases"
    print(f"For every additional 1 {row['Unit']}, resale price {direction} by ${

#  🔸 Town effects
print("\n📍 Town Effects on Resale Price (Compared to Base Town):\n")
town_mask = coef_df['Feature'].str.startswith('town_')
for _, row in coef_df[town_mask].iterrows():
    town_name = row['Feature'].replace('town_', '')
    coef = row['Coefficient']
    direction = "increases" if coef > 0 else "decreases"
    print(f"Living in {town_name} {direction} resale price by ${abs(coef):,.0f}
```

📈 Effect of Numerical Features:

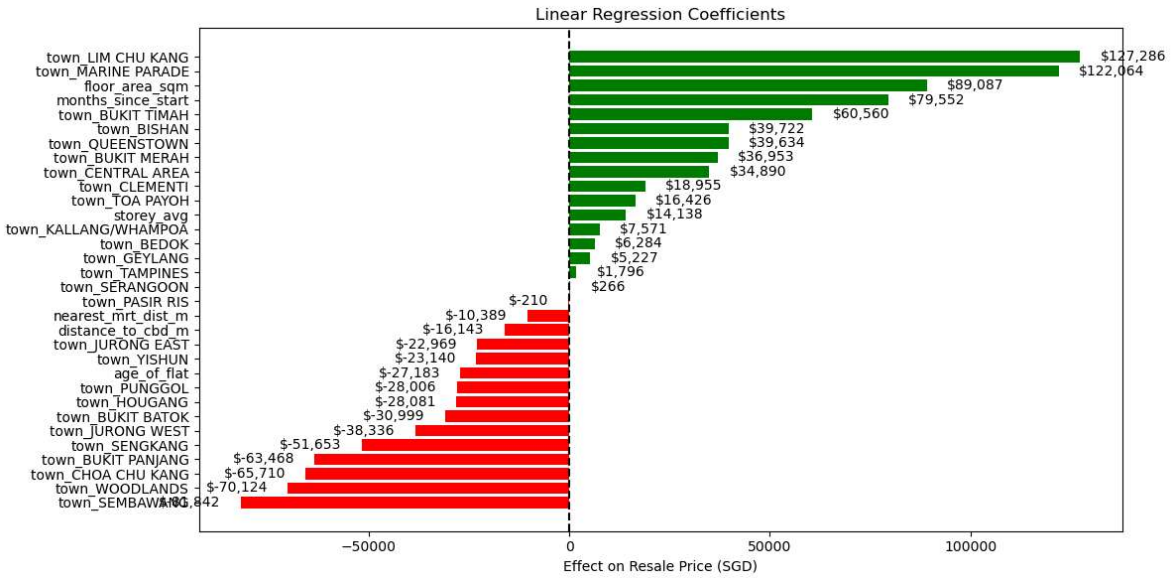
For every additional 1 sqm, resale price increases by $3,422 SGD.
For every additional 1 meter from MRT, resale price decreases by $26 SGD.
For every additional 1 meter from CBD, resale price decreases by $4 SGD.
For every additional 1 year of age, resale price decreases by $2,880 SGD.
For every additional 1 storey, resale price increases by $3,059 SGD.
For every additional 1 month (of transaction), resale price increases by $837 SGD.

📍 Town Effects on Resale Price (Compared to Base Town):

Living in BEDOK increases resale price by $6,284 SGD compared to the base town.
Living in BISHAN increases resale price by $39,722 SGD compared to the base town.
Living in BUKIT BATOK decreases resale price by $30,999 SGD compared to the base town.
Living in BUKIT MERAH increases resale price by $36,953 SGD compared to the base town.
Living in BUKIT PANJANG decreases resale price by $63,468 SGD compared to the base town.
Living in BUKIT TIMAH increases resale price by $60,560 SGD compared to the base town.
Living in CENTRAL AREA increases resale price by $34,890 SGD compared to the base town.
Living in CHOA CHU KANG decreases resale price by $65,710 SGD compared to the base town.
Living in CLEMENTI increases resale price by $18,955 SGD compared to the base town.
Living in GEYLANG increases resale price by $5,227 SGD compared to the base town.
Living in HOUGANG decreases resale price by $28,081 SGD compared to the base town.
Living in JURONG EAST decreases resale price by $22,969 SGD compared to the base town.
Living in JURONG WEST decreases resale price by $38,336 SGD compared to the base town.
Living in KALLANG/WHAMPOA increases resale price by $7,571 SGD compared to the base town.
Living in LIM CHU KANG increases resale price by $127,286 SGD compared to the base town.
Living in MARINE PARADE increases resale price by $122,064 SGD compared to the base town.
Living in PASIR RIS decreases resale price by $210 SGD compared to the base town.
Living in PUNGGOL decreases resale price by $28,006 SGD compared to the base town.
Living in QUEENSTOWN increases resale price by $39,634 SGD compared to the base town.
Living in SEMBAWANG decreases resale price by $81,842 SGD compared to the base town.
Living in SENGKANG decreases resale price by $51,653 SGD compared to the base town.
Living in SERANGOON increases resale price by $266 SGD compared to the base town.
Living in TAMPINES increases resale price by $1,796 SGD compared to the base town.
Living in TOA PAYOH increases resale price by $16,426 SGD compared to the base town.
Living in WOODLANDS decreases resale price by $70,124 SGD compared to the base town.
Living in YISHUN decreases resale price by $23,140 SGD compared to the base town.

In [114…

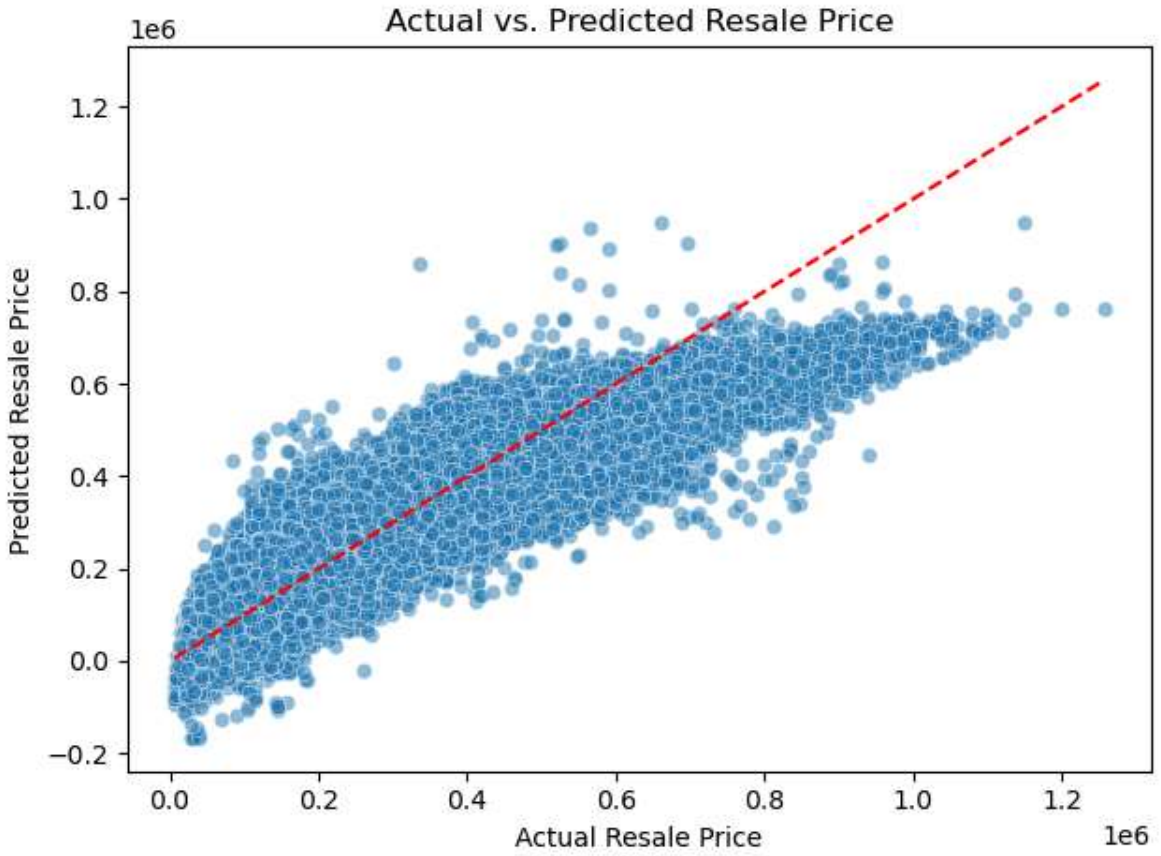
```
# Create DataFrame of coefficients
coef_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Coefficient': model1.coef_
})

# Add direction and color
coef_df['Direction'] = coef_df['Coefficient'].apply(lambda x: 'Increase' if x >
coef_df['Color'] = coef_df['Direction'].map({'Increase': 'green', 'Decrease': 'r

# Sort by actual coefficient value (ascending)
coef_df = coef_df.sort_values(by='Coefficient')

# Plot
plt.figure(figsize=(12, 6))
bars = plt.barh(coef_df['Feature'], coef_df['Coefficient'], color=coef_df['Color
plt.axvline(0, color='black', linestyle='--')
plt.xlabel("Effect on Resale Price (SGD)")
plt.title("Linear Regression Coefficients")

# Annotate each bar with SGD value
for bar, coef in zip(bars, coef_df['Coefficient']):
    width = bar.get_width()
    label_x = width + (5000 if coef > 0 else -5000)
    ha = 'left' if coef > 0 else 'right'
    plt.text(label_x, bar.get_y() + bar.get_height() / 2,
             f"${coef:,.0f}", va='center', ha=ha, color='black')

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

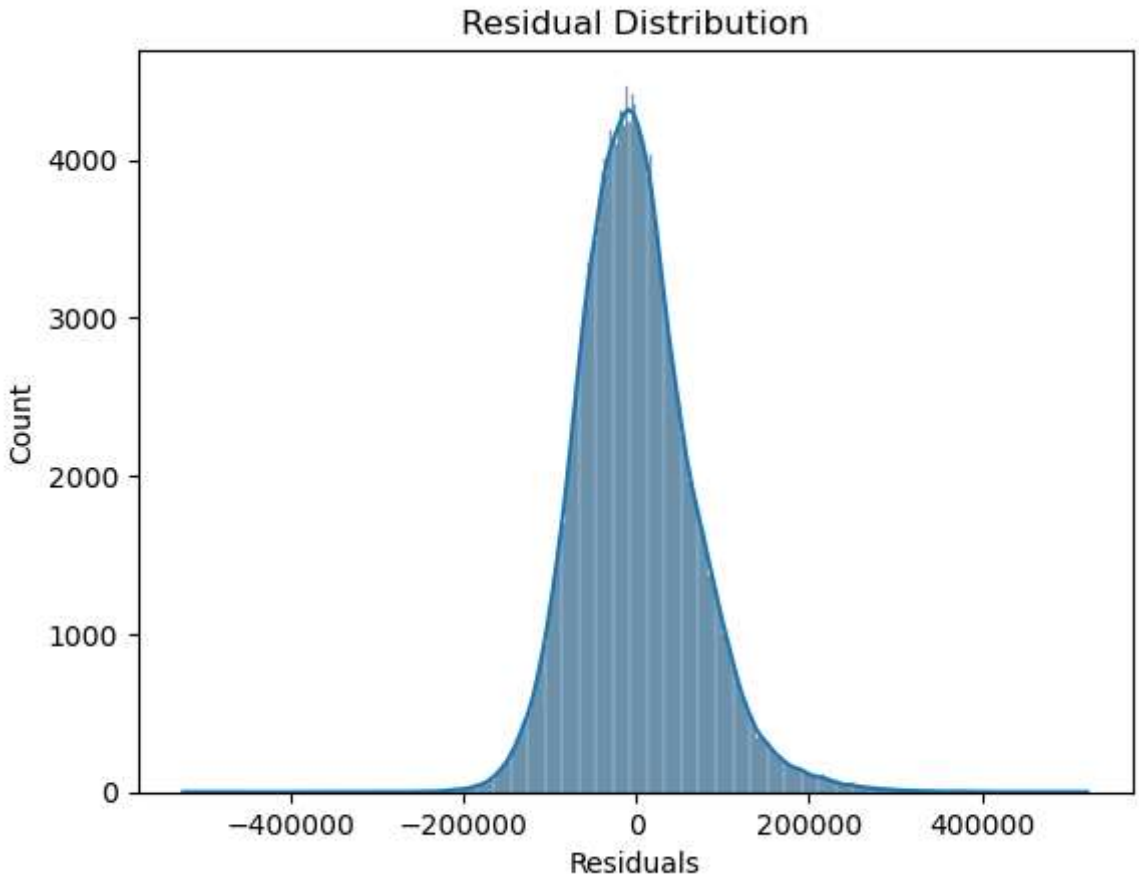


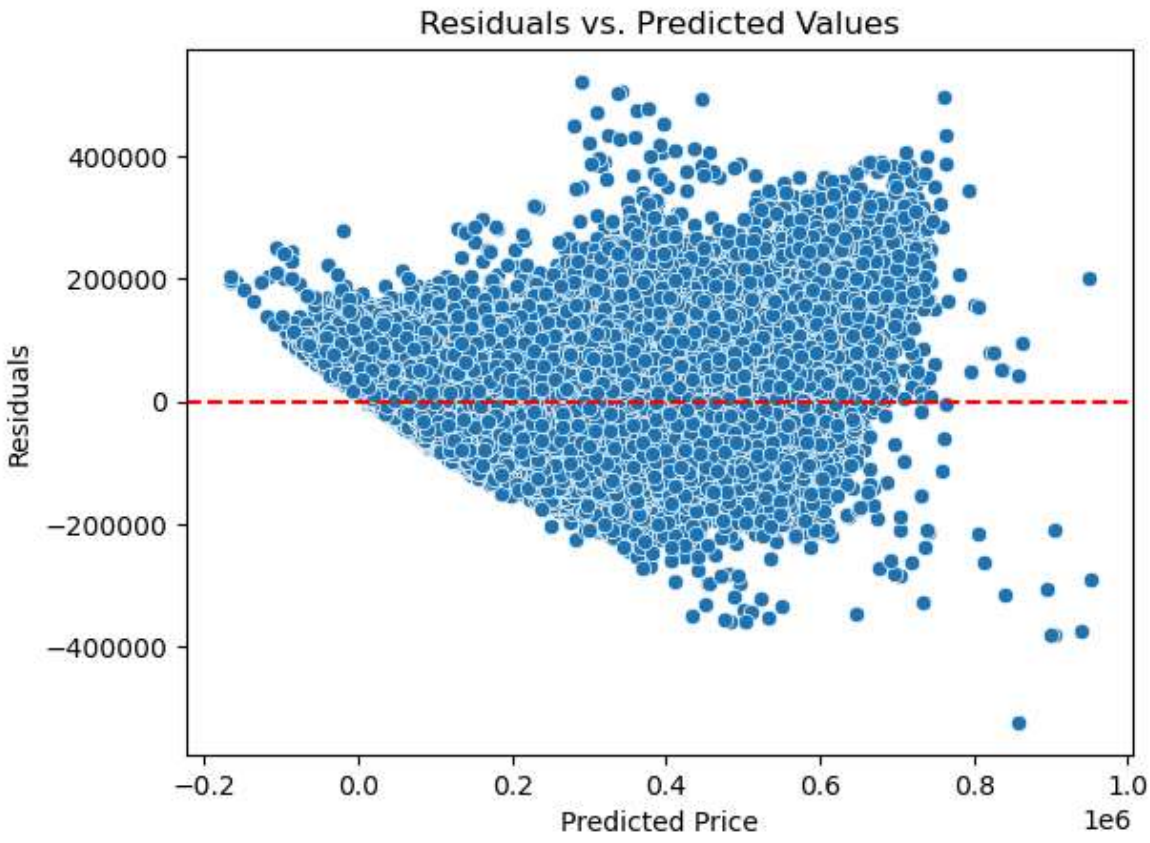
In [116…

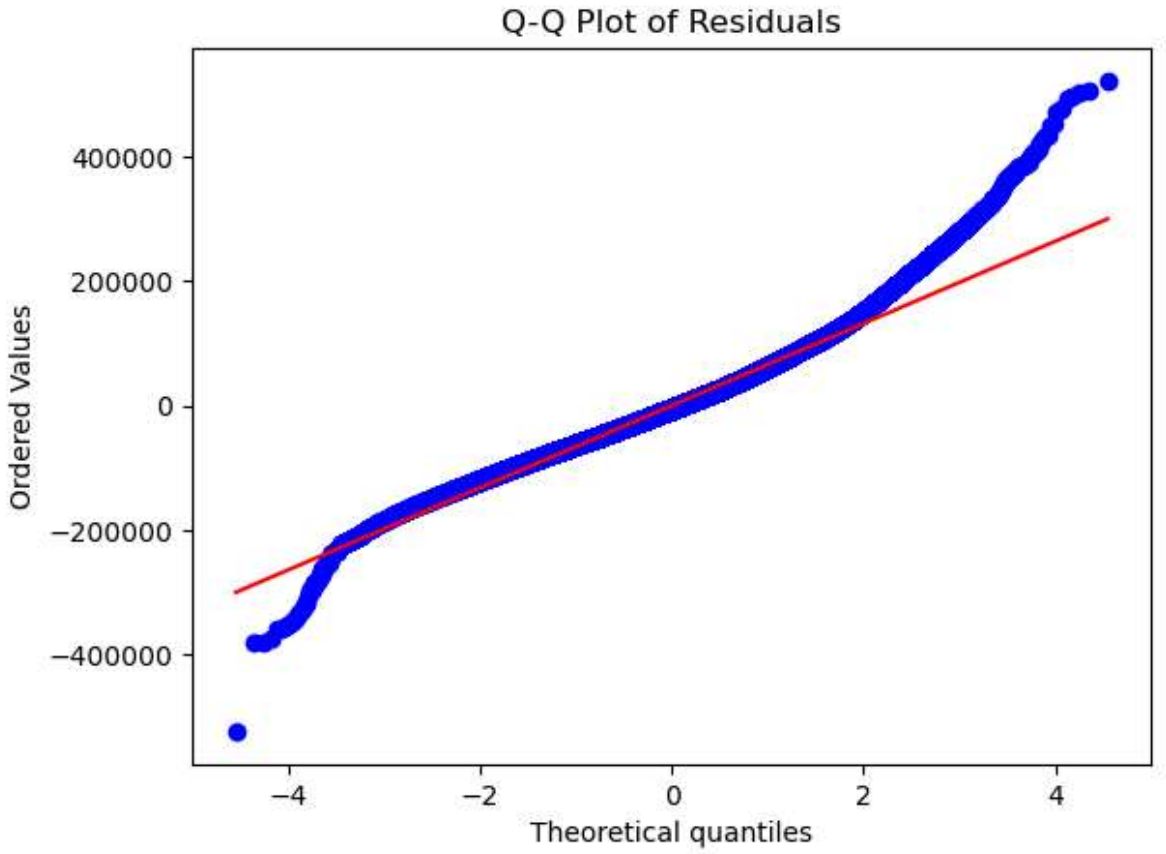
```
import scipy.stats as stats

y_pred = model1.predict(X_test)
residuals = y_test - y_pred

sns.scatterplot(x=y_test, y=y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')  # p
plt.xlabel("Actual Resale Price")
plt.ylabel("Predicted Resale Price")
plt.title("Actual vs. Predicted Resale Price")
```

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

# Histogram of residuals
sns.histplot(residuals, kde=True)
plt.title("Residual Distribution")
plt.xlabel("Residuals")
plt.show()

# Residuals vs. predicted
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(0, color='red', linestyle='--')
plt.title("Residuals vs. Predicted Values")
plt.xlabel("Predicted Price")
plt.ylabel("Residuals")
plt.show()

stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals")
plt.show()
```




Residual Distribution
Count
Residuals
4000
3000
2000
1000
0
−400000
−200000
0
200000
400000


Residuals vs. Predicted Values
Residuals
Predicted Price
1e6
400000
200000
0
−200000
−400000
−0.2
0.0
0.2
0.4
0.6
0.8
1.0



In [117…

```
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Adjusted R²
n = X_test.shape[0]  # rows
k = X_test.shape[1]  # predictors
adj_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)

# Print results
print(f"R²: {r2:.4f}")
print(f"Adjusted R²: {adj_r2:.4f}")
print(f"RMSE: {rmse:,.0f} SGD")
print(f"MAE: {mae:,.0f} SGD")
```

```
R²: 0.7982
Adjusted R²: 0.7981
RMSE: 66,673 SGD
MAE: 51,324 SGD
```

In [118…

```
### Test prediction ability of LR

test_input_dict = {
    'floor_area_sqm': 149,
    'nearest_mrt_dist_m': 400,
    'distance_to_cbd_m': 13000,
    'age_of_flat': 25,
    'storey_avg': 8,
    'months_since_start': 100,
    'town_BEDOK': 0,
    'town_BISHAN': 0,
    'town_BUKIT BATOK': 0,
```

```
    'town_BUKIT MERAH': 0,
    'town_BUKIT PANJANG': 0,
    'town_BUKIT TIMAH': 0,
    'town_CENTRAL AREA': 0,
    'town_CHOA CHU KANG': 0,
    'town_CLEMENTI': 0,
    'town_GEYLANG': 0,
    'town_HOUGANG': 0,
    'town_JURONG EAST': 0,
    'town_JURONG WEST': 0,
    'town_KALLANG/WHAMPOA': 0,
    'town_LIM CHU KANG': 0,
    'town_MARINE PARADE': 0,
    'town_PASIR RIS': 0,
    'town_PUNGGOL': 0,
    'town_QUEENSTOWN': 0,
    'town_SEMBAWANG': 0,
    'town_SENGKANG': 0,
    'town_SERANGOON': 0,
    'town_TAMPINES': 1,
    'town_TOA PAYOH': 0,
    'town_WOODLANDS': 0,
    'town_YISHUN': 0
}
test_input = pd.DataFrame([test_input_dict])


# If you used a StandardScaler for numerical features:
test_input[num_features] = scaler.transform(test_input[num_features])
```

In [119…

```
predicted_price = model1.predict(test_input)[0]
print(f"Predicted resale price: ${predicted_price:,.0f} SGD")
```

```
Predicted resale price: $453,495 SGD
```

In [120…

```
import plotly.express as px

# Ensure 'month' is datetime and extract 'year'
geo_df_MRT_cbd['month'] = pd.to_datetime(geo_df_MRT_cbd['month'])
geo_df_MRT_cbd['year'] = geo_df_MRT_cbd['month'].dt.year
geo_df_MRT_cbd = geo_df_MRT_cbd.sort_values('year')

# Calculate global max values for fixed axis range
xmax = geo_df_MRT_cbd['resale_price'].max()
ymax = 6000

# Group by year to compute total transactions
yearly_counts = geo_df_MRT_cbd.groupby("year").size().reset_index(name="count")

# Create animated histogram
fig = px.histogram(
    geo_df_MRT_cbd,
    x="resale_price",
    color="flat_type",
    animation_frame="year",
    nbins=50,
    barmode='overlay',
    color_discrete_sequence=px.colors.qualitative.Set2
)
```

```
# Fix axes for consistency across frames
fig.update_layout(
    xaxis=dict(range=[0, geo_df_MRT_cbd['resale_price'].max()]),
     yaxis=dict(range=[0, ymax]),
    yaxis_title="Count",
    xaxis_title="Resale Price (SGD)",
    title_x=0.5
)

# Add annotation dynamically for each animation frame
for frame in fig.frames:
    year = int(frame.name)
    count = int(yearly_counts.loc[yearly_counts['year'] == year, 'count'].values

    frame.layout.annotations = [
        go.layout.Annotation(
            text=f"Total Transactions: {count:,}",
            x=0.5,
            y=0.95,
            xref="paper",
            yref="paper",
            showarrow=False,
            font=dict(size=16, color="black")
        )
    ]

# Also show annotation on initial frame
initial_year = int(fig.frames[0].name)
initial_count = int(yearly_counts.loc[yearly_counts['year'] == initial_year, 'co
fig.update_layout(
    annotations=[
        go.layout.Annotation(
            text=f"Total Transactions: {initial_count:,}",
            x=0.5,
            y=0.95,
            xref="paper",
            yref="paper",
            showarrow=False,
            font=dict(size=16, color="black")
        )
    ]
)

fig.show()
```

DISCUSSION: Our Baseline Linear Regression model gives :

-> R^2=0.7982, RMSE=66,673 SGD and MA= 51,324 SG. From the EDA, and residual plot in model1n, we can see that there is non-linearity in the dat.

-> We will tackle this non-linearity in our model 2 as well as data points that have high leverage and influence to improve our model (to remove high leverage data points after train-test split).. Eg. We can try to improve the model by log-transformation specifically box-cox of the resale_price data with cross validation to find the best lambda value.

MODEL 2- LINEAR REGRESSION WITH BOX-COX TRANSFORMATION OF 'RESALE PRICE' AND REMOVAL OF NUMERICAL OUTLIERS (HIGH LEVERAGE AND INFLUENCE), AND REMOVAL OF CATEGORICAL OUTLIERS (FLAT_TYPE =[1 ROOM, MULTI-GENERATION], TOWN=['LIM CHU KANG', 'BUKIT TIMAH']

In [128…

```
df_model2= geo_df_MRT_cbd.copy()
df_model2
```

Out[128…

	month	town	flat_type	block	street_name	storey_range	floor_area_sqm
0	1990-01-01	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	10 TO 12	31.0
8279	1990-01-08	JURONG WEST	EXECUTIVE	558	JURONG WEST ST 42	10 TO 12	148.0
8280	1990-01-08	JURONG WEST	EXECUTIVE	472	JURONG WEST ST 41	10 TO 12	151.0
8281	1990-01-08	JURONG WEST	EXECUTIVE	472	JURONG WEST ST 41	10 TO 12	151.0
8282	1990-01-08	JURONG WEST	EXECUTIVE	472	JURONG WEST ST 41	10 TO 12	149.0
...	...	...	...	...	...	...	...
762048	2020-01-03	YISHUN	4 ROOM	458	YISHUN AVE 11	07 TO 09	93.0
762049	2020-01-03	YISHUN	4 ROOM	458	YISHUN AVE 11	10 TO 12	93.0
762050	2020-01-03	YISHUN	4 ROOM	416	YISHUN AVE 11	04 TO 06	104.0
762037	2020-01-03	YISHUN	4 ROOM	431D	YISHUN AVE 1	04 TO 06	92.0
766098	2020-01-07	CHOA CHU KANG	EXECUTIVE	526	CHOA CHU KANG ST 51	04 TO 06	142.0

824732 rows × 20 columns

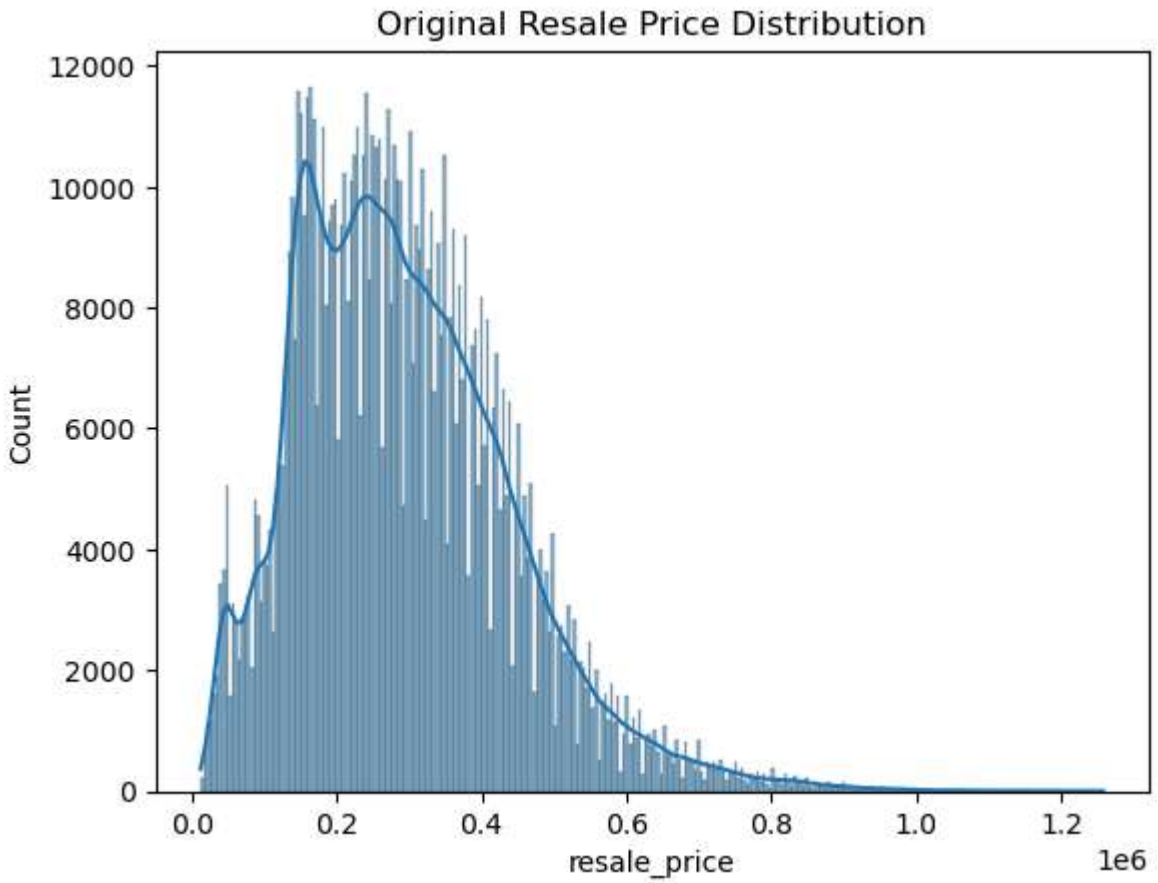
In [130…

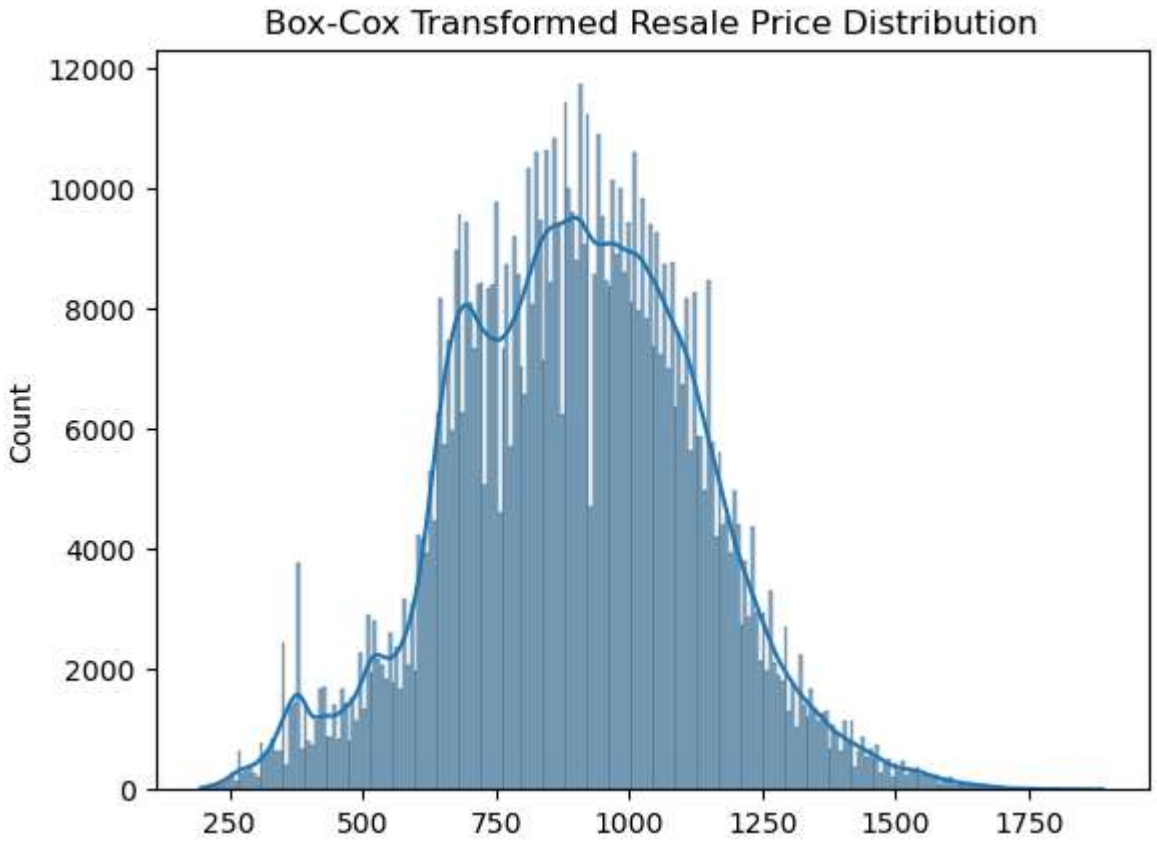
```
#Further pre-processing to remove irrelevant feature columns and translate other

###model 2 specific removal of outliers
#remove lim chu kang and bukit timah for very low counts
df_model2 = df_model2[~df_model2['town'].isin(['LIM CHU KANG', 'BUKIT TIMAH'])]

#drop flat-type = 1room and multi gen for very low counts
df_model2 = df_model2[~df_model2['flat_type'].isin(['1 ROOM', 'MULTI-GENERATION'
```

In [131…

```
#box cox transformation of y='resale_price' with optimal lambda
y = df_model2['resale_price']
y_transformed, fitted_lambda = boxcox(y)

# Add it back to dataframe
df_model2['resale_price_boxcox'] = y_transformed
```

```
# Optionally inspect distribution
sns.histplot(y, kde=True)
plt.title("Original Resale Price Distribution")
plt.show()

sns.histplot(y_transformed, kde=True)
plt.title("Box-Cox Transformed Resale Price Distribution")
plt.show()

print(f"Optimal lambda: {fitted_lambda:.4f}")
```






```
Optimal lambda: 0.4858
```

In [132…

```
#get age of flat
current_year=2020
df_model2['age_of_flat']= current_year-df_model2['lease_commence_date']

#convert storey_range to numerical variable
def parse_storey_range(s):
    try:
        low, high = map(int, s.split(' TO '))
        return (low + high) / 2
    except:
        return None

df_model2['storey_avg'] = df_model2['storey_range'].apply(parse_storey_range)
df_model2.drop(columns=['storey_range'], axis=1, inplace=True)
df_model2.drop(columns=['block','street_name','flat_model','year','latitude','lo

#handle the 'month' variable to be continuous and numerical for LR model to unde
df_model2['month'] = pd.to_datetime(df_model2['month'])
df_model2['months_since_start'] = (
    df_model2['month'].dt.to_period('M') - df_model2['month'].min().to_period('M
).apply(lambda x: x.n)
df_model2.drop(columns=['month'], axis=1, inplace=True)
```

In [133…

```
df_model2
```

Out[133…

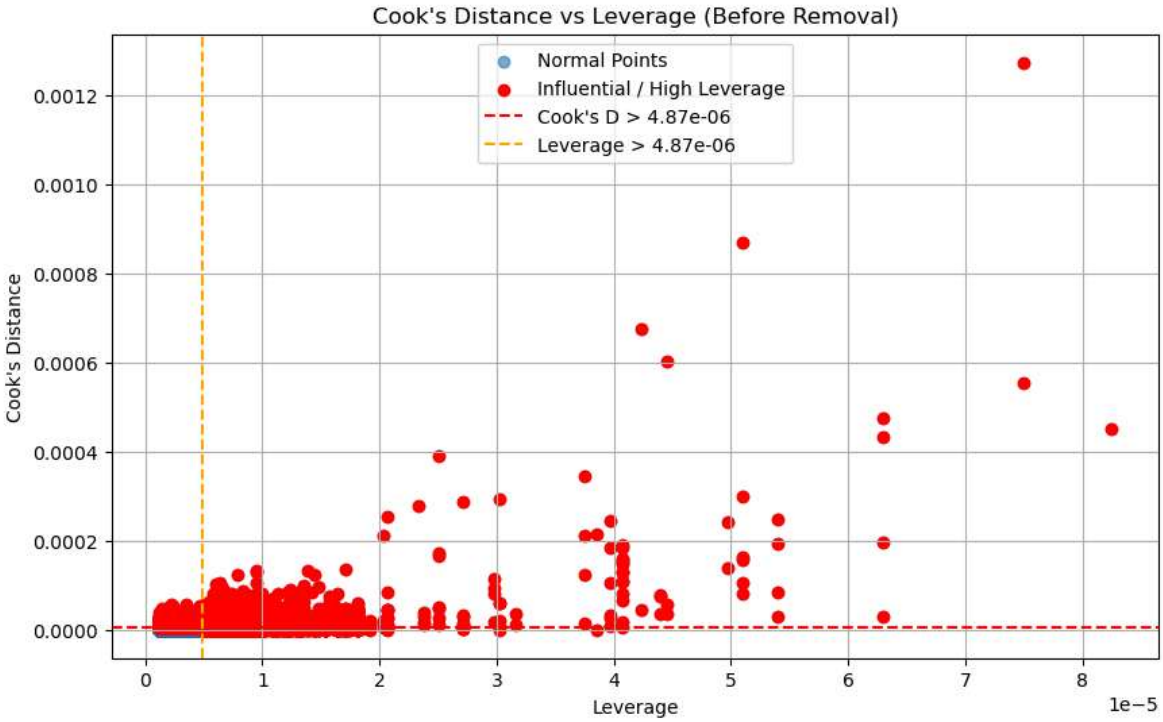
	town	floor_area_sqm	resale_price	nearest_mrt_dist_m	distance_to_cbd_m
8279	JURONG WEST	148.0	160000.0	1181.168902	13929.65656
8280	JURONG WEST	151.0	150000.0	431.211489	13145.27717
8281	JURONG WEST	151.0	159600.0	431.211489	13145.27717
8282	JURONG WEST	149.0	193000.0	431.211489	13145.27717
8283	JURONG WEST	150.0	159600.0	1159.142003	12985.58855
…	…	…	…	…	…
762048	YISHUN	93.0	370000.0	1476.901513	12892.07836
762049	YISHUN	93.0	400888.0	1476.901513	12892.07836
762050	YISHUN	104.0	345000.0	1334.093222	13227.14071
762037	YISHUN	92.0	360000.0	1546.425808	12825.76121
766098	CHOA CHU KANG	142.0	470000.0	676.296577	13897.31130

820642 rows × 9 columns

In [135…

```
#use cook's distance to remove high-leverage and influential numerical outliers
import statsmodels.api as sm

# Step 1: Prepare X and y
X = df_model2[['floor_area_sqm']].copy()
X = sm.add_constant(X)  # Add intercept
y = df_model2['resale_price']

# Step 2: Fit linear regression
model = sm.OLS(y, X).fit()

# Step 3: Get influence metrics
influence = model.get_influence()
cooks_d, _ = influence.cooks_distance
leverage = influence.hat_matrix_diag

# Step 4: Add metrics to dataframe
df_plot = df_model2.copy()
df_plot['cooks_d'] = cooks_d
df_plot['leverage'] = leverage

# Step 5: Define thresholds
n = len(df_plot)
p = X.shape[1]
cook_thresh = 4 / n
lev_thresh = 2 * p / n
```

```
# Step 6: Plot Leverage vs Cook's Distance
plt.figure(figsize=(10, 6))
plt.scatter(df_plot['leverage'], df_plot['cooks_d'], alpha=0.6, label='Normal Po

# Highlight points beyond threshold in red
mask = (df_plot['cooks_d'] > cook_thresh) | (df_plot['leverage'] > lev_thresh)
plt.scatter(df_plot.loc[mask, 'leverage'],
            df_plot.loc[mask, 'cooks_d'],
            color='red', label='Influential / High Leverage')

plt.axhline(y=cook_thresh, color='red', linestyle='--', label=f"Cook's D > {cook
plt.axvline(x=lev_thresh, color='orange', linestyle='--', label=f'Leverage > {le
plt.xlabel('Leverage')
plt.ylabel("Cook's Distance")
plt.title("Cook's Distance vs Leverage (Before Removal)")
plt.legend()
plt.grid(True)
plt.show()

# Step 7: Print those data points beyond thresholds
print("\n📌 Influential or High-Leverage Points:")
display(df_plot.loc[mask, ['floor_area_sqm', 'resale_price', 'cooks_d', 'leverag
```



📌 Influential or High-Leverage Points:

	floor_area_sqm	resale_price	cooks_d	leverage
69266	297.0	335000.0	1.273196e-03	0.000075
31347	261.0	205000.0	8.700372e-04	0.000051
32568	246.0	175000.0	6.738638e-04	0.000042
52227	250.0	240000.0	6.033494e-04	0.000045
370096	297.0	565000.0	5.539083e-04	0.000075
...	...	...	...	...
618125	155.0	505000.0	1.700019e-14	0.000008
556888	155.0	505000.0	1.700019e-14	0.000008
590796	155.0	505000.0	1.700019e-14	0.000008
569989	155.0	505000.0	1.700019e-14	0.000008
294600	155.0	505000.0	1.700019e-14	0.000008

90488 rows × 4 columns

In [136…
```
outlier_indices = df_model2[mask].index
```

In [137…
```
outliers_in_original_df = df_model2.loc[outlier_indices]
outliers_in_original_df
```

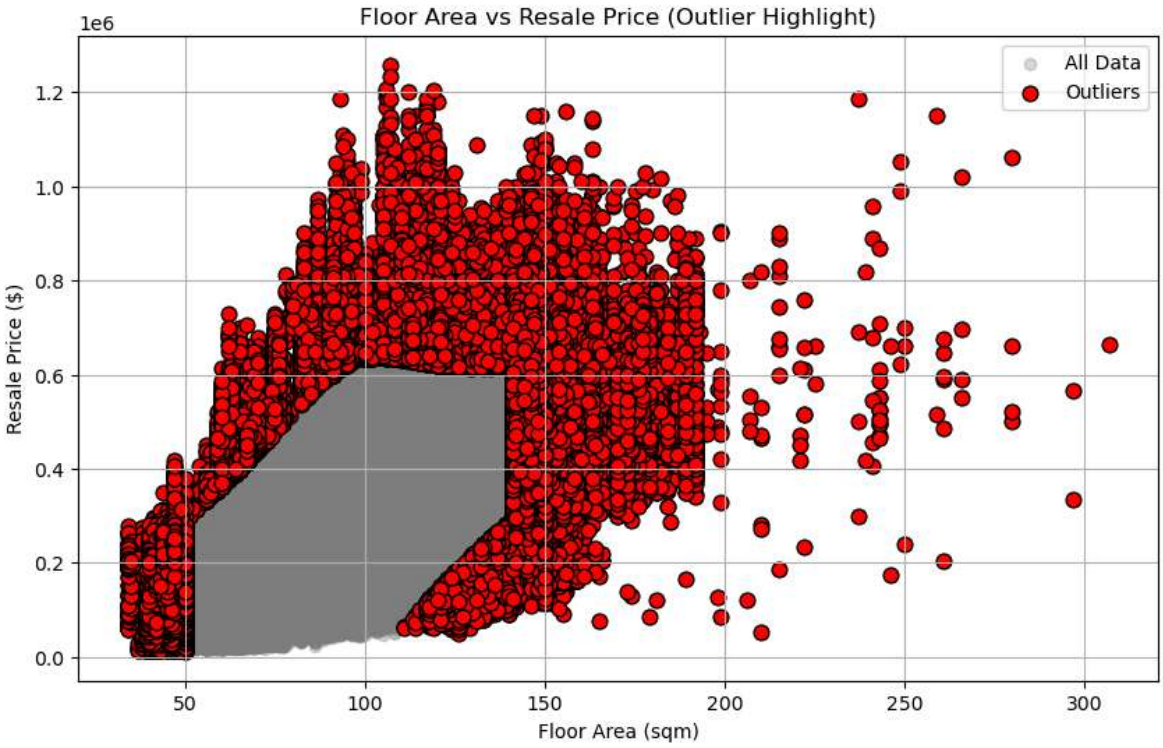
Out[137…

	town	floor_area_sqm	resale_price	nearest_mrt_dist_m	distance_to_cbd
8279	JURONG WEST	148.0	160000.0	1181.168902	13929.656
8280	JURONG WEST	151.0	150000.0	431.211489	13145.277
8281	JURONG WEST	151.0	159600.0	431.211489	13145.277
8282	JURONG WEST	149.0	193000.0	431.211489	13145.277
8283	JURONG WEST	150.0	159600.0	1159.142003	12985.588
…	…	…	…	…	
761981	YISHUN	47.0	225000.0	1601.334985	13207.756
761982	YISHUN	47.0	225000.0	1601.334985	13207.756
761983	YISHUN	47.0	250000.0	1601.334985	13207.756
761974	WOODLANDS	177.0	675000.0	491.453426	15161.491
766098	CHOA CHU KANG	142.0	470000.0	676.296577	13897.311

90488 rows × 9 columns

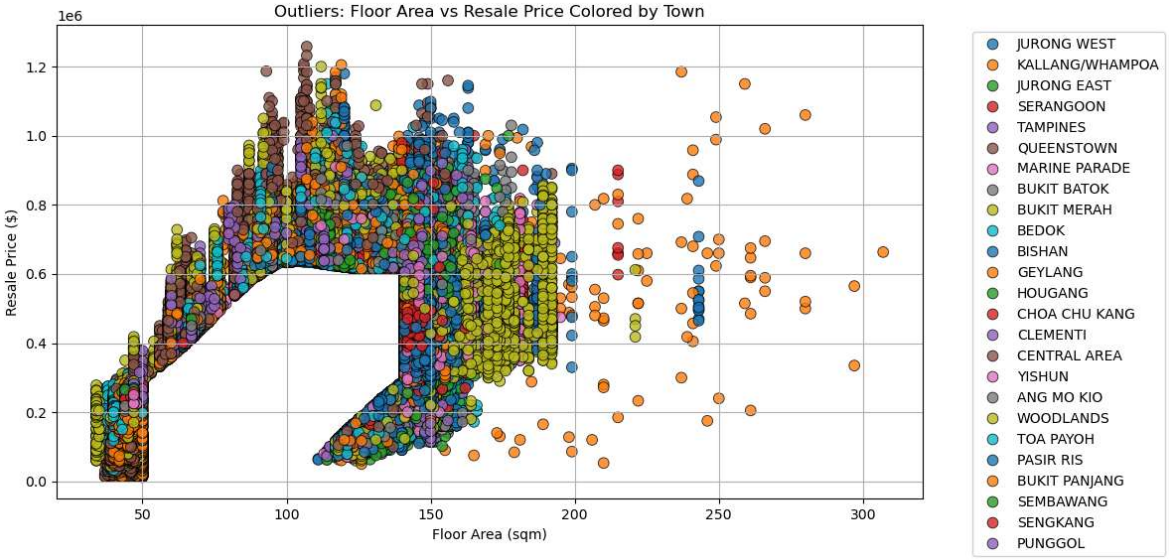
In [138…

```
plt.figure(figsize=(10,6))

# Plot all points in gray
plt.scatter(df_model2['floor_area_sqm'], df_model2['resale_price'],
            alpha=0.3, label='All Data', color='gray')

# Highlight outliers in red
plt.scatter(outliers_in_original_df['floor_area_sqm'], outliers_in_original_df['
            color='red', label='Outliers', edgecolor='black', s=60)

plt.xlabel('Floor Area (sqm)')
plt.ylabel('Resale Price ($)')
plt.title('Floor Area vs Resale Price (Outlier Highlight)')
plt.legend()
plt.grid(True)
plt.show()
```



In [139…

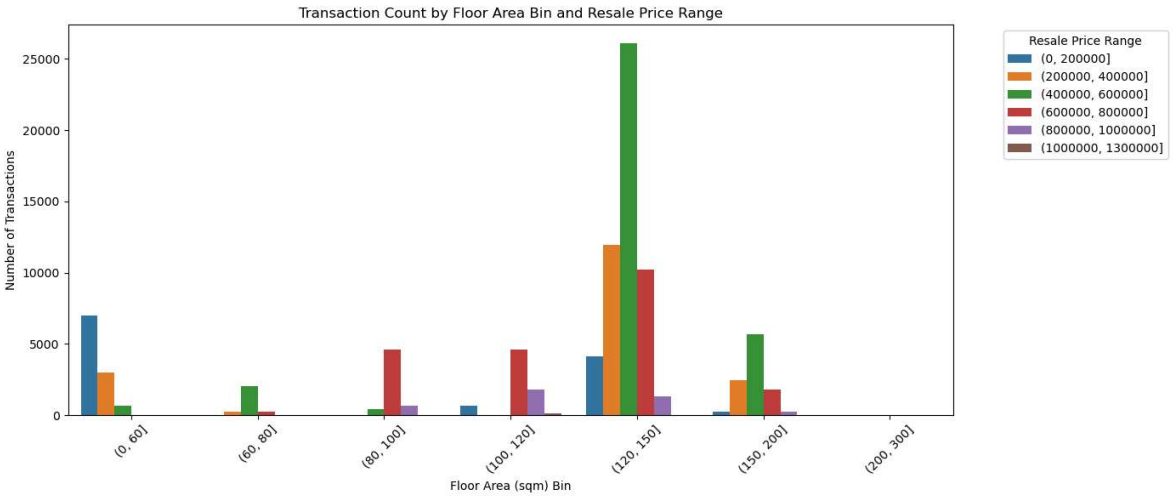
```
plt.figure(figsize=(12, 6))
sns.scatterplot(
    data=outliers_in_original_df,
    x='floor_area_sqm',
    y='resale_price',
    hue='town',
    palette='tab10',  # or 'Set2', 'tab20', etc.
    s=60,
    edgecolor='black',
    alpha=0.8
)

plt.xlabel('Floor Area (sqm)')
plt.ylabel('Resale Price ($)')
plt.title('Outliers: Floor Area vs Resale Price Colored by Town')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [140…

```
# Bin floor area and resale price
df_binned = outliers_in_original_df.copy()
df_binned['area_bin'] = pd.cut(df_binned['floor_area_sqm'], bins=[0, 60, 80, 100
df_binned['price_bin'] = pd.cut(df_binned['resale_price'], bins=[0, 200000, 4000

# Count combinations
count_df = df_binned.groupby(['area_bin', 'price_bin']).size().reset_index(name=

# Plot grouped bar chart
plt.figure(figsize=(14, 6))
sns.barplot(data=count_df, x='area_bin', y='count', hue='price_bin')
plt.title('Transaction Count by Floor Area Bin and Resale Price Range')
plt.xlabel('Floor Area (sqm) Bin')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.legend(title='Resale Price Range', bbox_to_anchor=(1.05, 1), loc='upper left
plt.tight_layout()
plt.show()
```

```
C:\Users\bulle\AppData\Local\Temp\ipykernel_18112\2594423229.py:7: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a futu
re version of pandas. Pass observed=False to retain current behavior or observed=
True to adopt the future default and silence this warning.
```



There are outliers mainly for very small or very large HDB units (not representative of the usual HDB layouts for eg. Terraces etc.)

In [142…

```
#remove the data points with high cook distance
df_model2_cleaned = df_plot[~mask].copy()

# Define X and y again using the cleaned data
X_clean = df_model2_cleaned.drop(columns=['resale_price', 'resale_price_boxcox',
X_clean = pd.get_dummies(X_clean, drop_first=True)
y_clean = df_model2_cleaned['resale_price_boxcox']
```

In [143…

```
X_clean
```

Out[143...

	floor_area_sqm	nearest_mrt_dist_m	distance_to_cbd_m	age_of_flat	storey_avg
8288	59.0	378.038350	3744.729377	44	11.0
8290	60.0	215.870279	3381.455518	42	14.0
8291	66.0	741.970038	3083.176414	40	8.0
8292	60.0	481.621033	3611.702267	34	11.0
8293	60.0	376.385597	3151.547368	39	5.0
...	...	...	...	...	...
762047	104.0	1444.291949	13036.891630	32	8.0
762048	93.0	1476.901513	12892.078360	7	8.0
762049	93.0	1476.901513	12892.078360	7	11.0
762050	104.0	1334.093222	13227.140710	27	5.0
762037	92.0	1546.425808	12825.761210	5	5.0

730154 rows × 30 columns

Lets try running our Linear Regression Model

In [145...

```
## to standardise only after train/test split, and to standardise on training da

X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(X_clean, y_clean, te

num_features = ['floor_area_sqm', 'nearest_mrt_dist_m', 'distance_to_cbd_m',
                'age_of_flat', 'storey_avg', 'months_since_start']
scaler2 = StandardScaler()
X_train_2[num_features] = scaler2.fit_transform(X_train_2[num_features])
X_test_2[num_features] = scaler2.transform(X_test_2[num_features])  # use same s
```

In [146...

```
model2 = LinearRegression()
model2.fit(X_train_2, y_train_2)
```

Out[146...

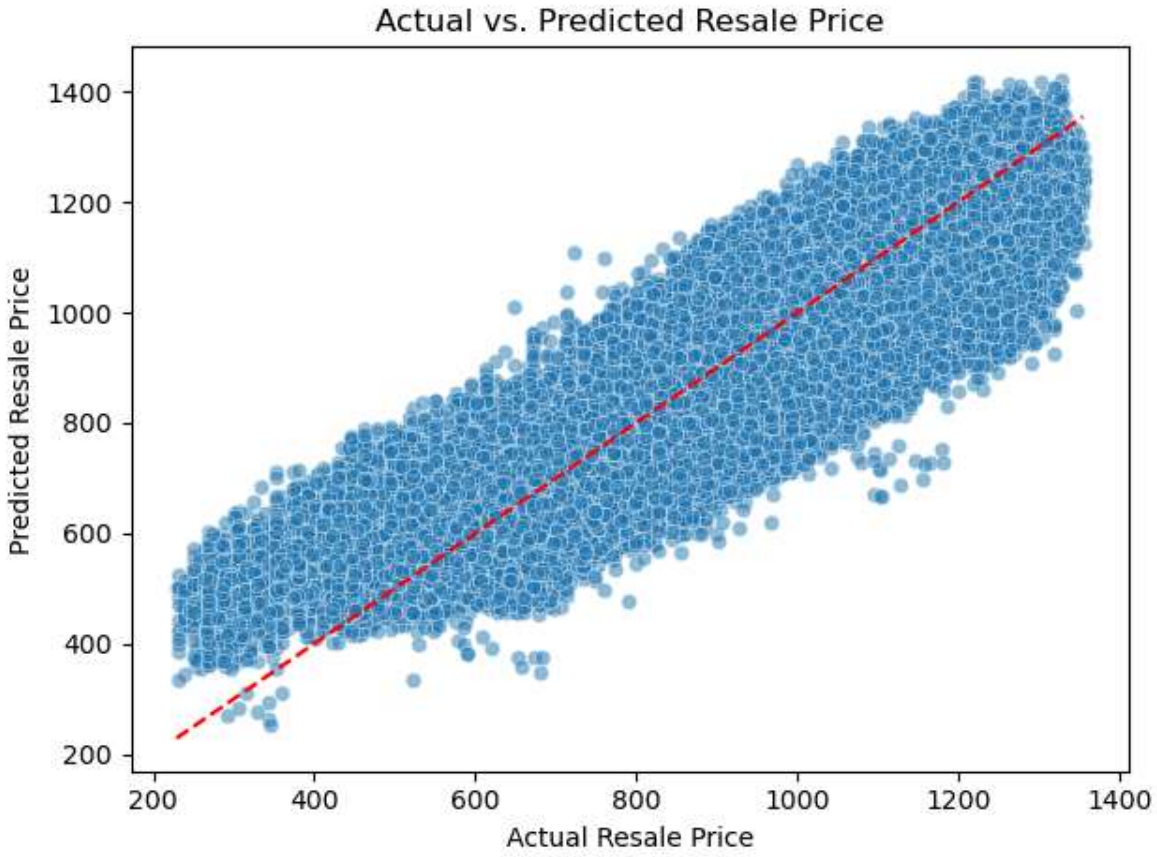
LinearRegression

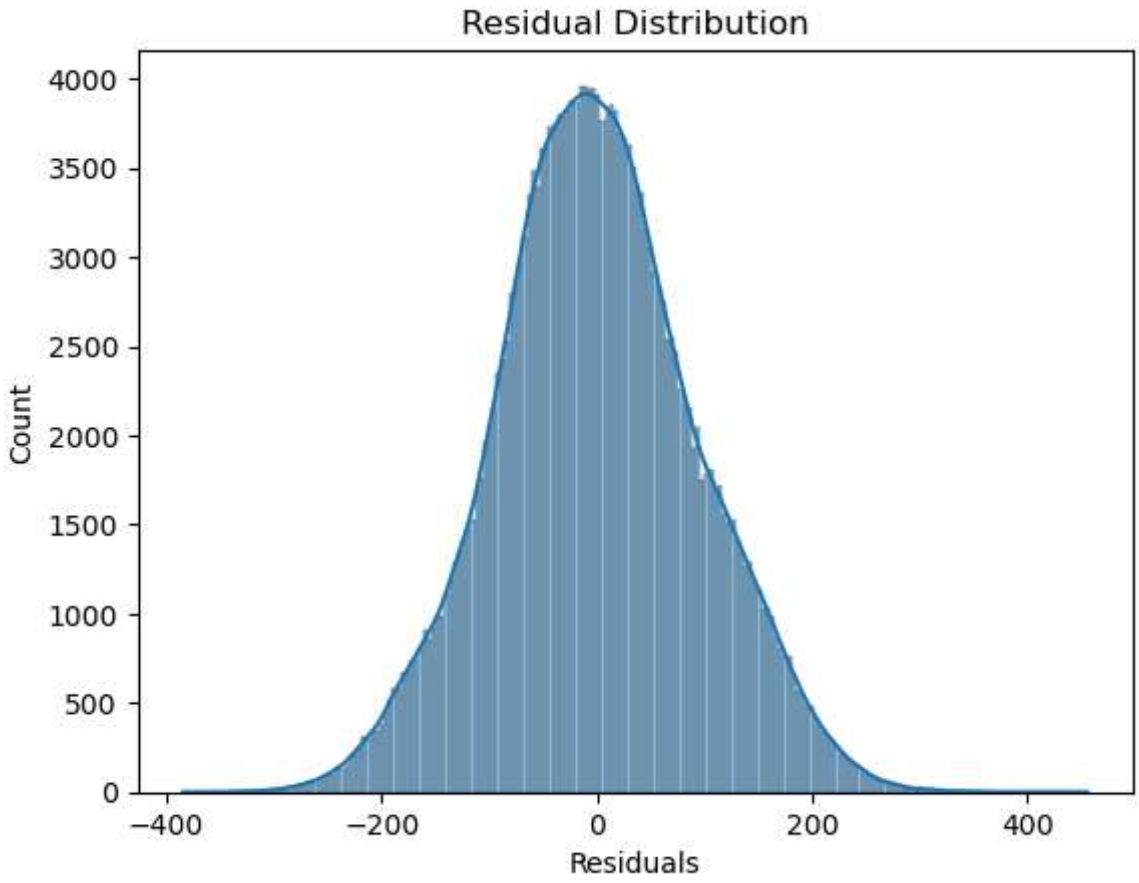
LinearRegression()

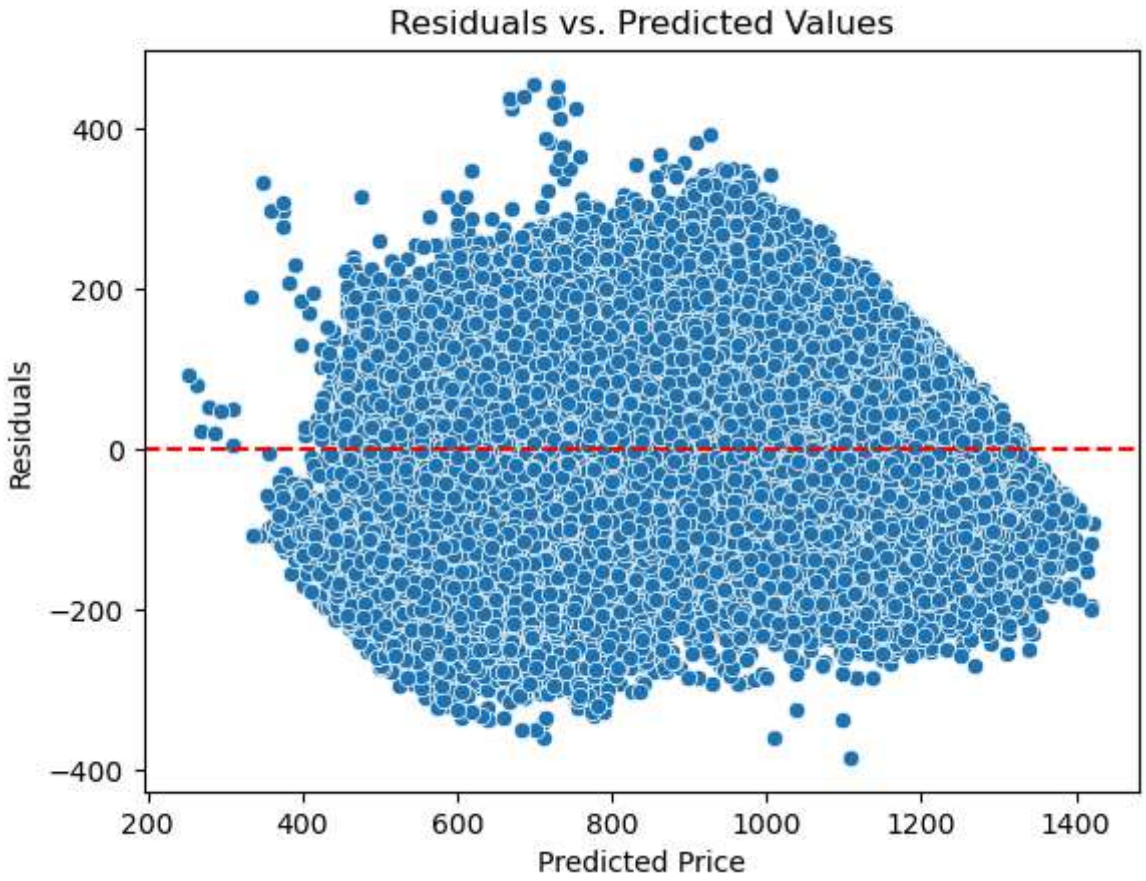
In [147...

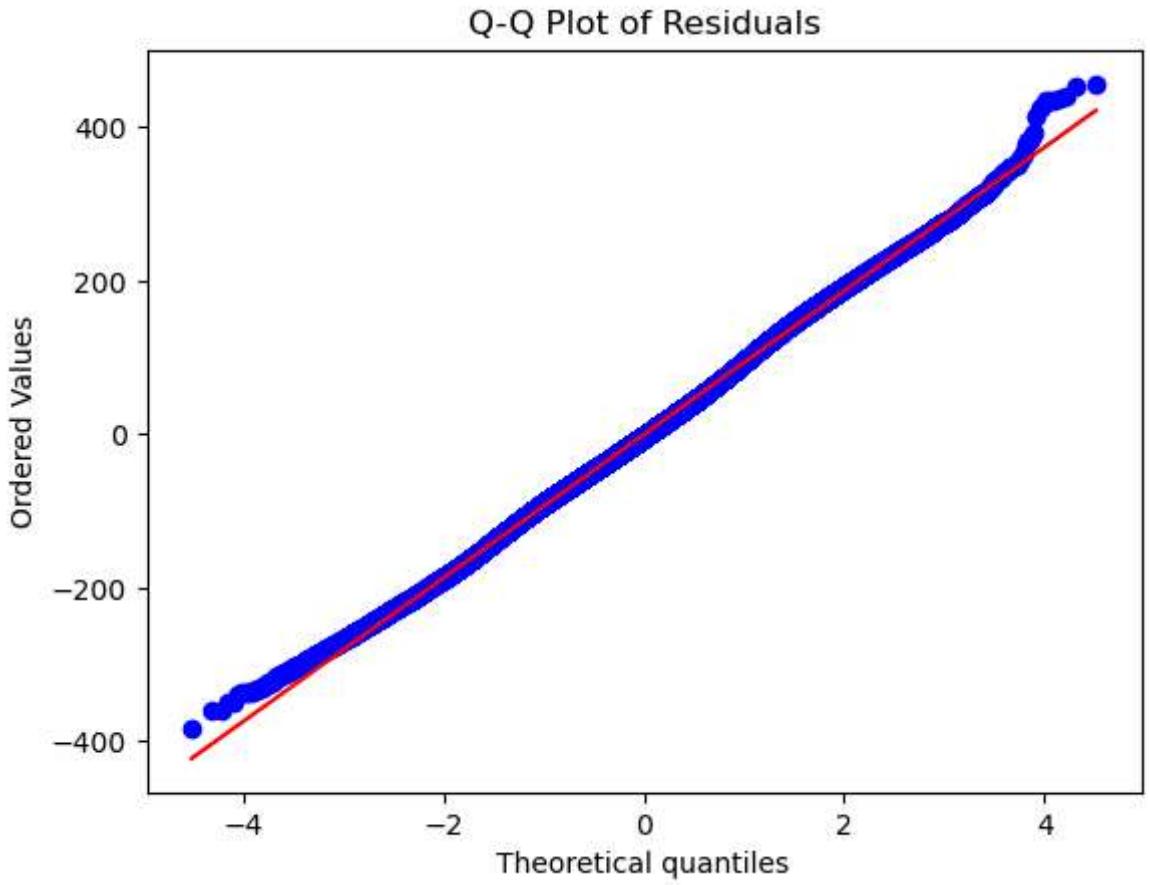
```
import scipy.stats as stats

y_pred_2 = model2.predict(X_test_2)
residuals_2 = y_test_2 - y_pred_2

sns.scatterplot(x=y_test_2, y=y_pred_2, alpha=0.5)
plt.plot([y_test_2.min(), y_test_2.max()], [y_test_2.min(), y_test_2.max()], 'r-
plt.xlabel("Actual Resale Price")
plt.ylabel("Predicted Resale Price")
plt.title("Actual vs. Predicted Resale Price")
plt.tight_layout()
```

```
plt.show()

# Histogram of residuals
sns.histplot(residuals_2, kde=True)
plt.title("Residual Distribution")
plt.xlabel("Residuals")
plt.show()

# Residuals vs. predicted
sns.scatterplot(x=y_pred_2, y=residuals_2)
plt.axhline(0, color='red', linestyle='--')
plt.title("Residuals vs. Predicted Values")
plt.xlabel("Predicted Price")
plt.ylabel("Residuals")
plt.show()

stats.probplot(residuals_2, dist="norm", plot=plt)
plt.title("Q-Q Plot of Residuals")
plt.show()
```




Residual Distribution
Count
Residuals
4000
3500
3000
2500
2000
1500
1000
500
0
−400
−200
0
200
400


Residuals vs. Predicted Values
Residuals
Predicted Price
400
200
0
−200
−400
200
400
600
800
1000
1200
1400

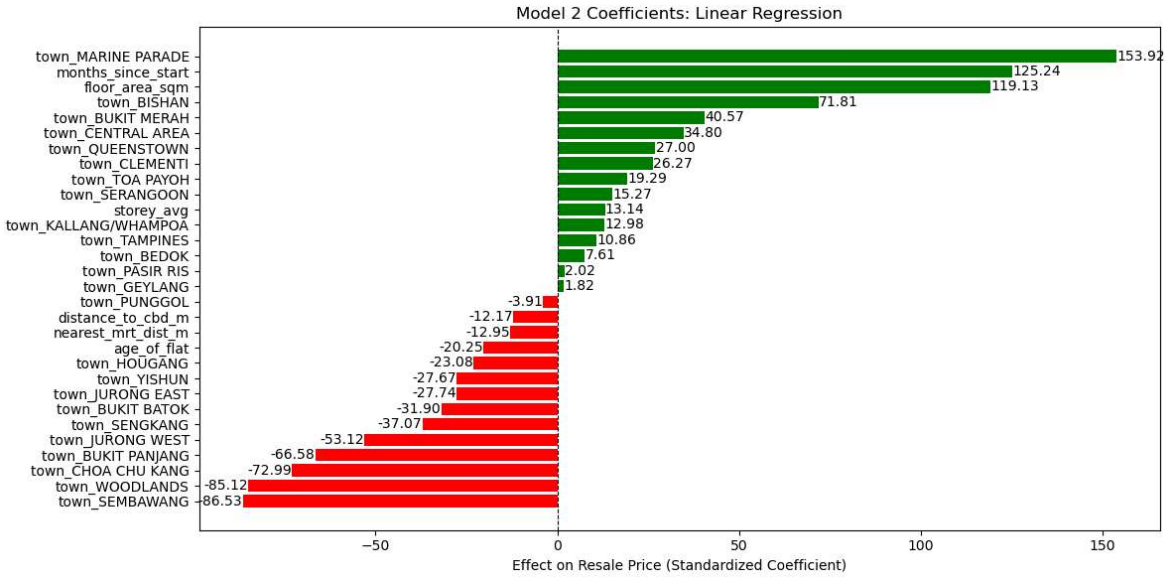


In [148…
```
mse = mean_squared_error(y_test_2, y_pred_2)
rmse = mse ** 0.5
mae = mean_absolute_error(y_test_2, y_pred_2)
r2 = r2_score(y_test_2, y_pred_2)

# Adjusted R²
n = X_test_2.shape[0]  # rows
k = X_test_2.shape[1]  # predictors
adj_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)

# Print results
print(f"R²: {r2:.4f}")
print(f"Adjusted R²: {adj_r2:.4f}")
print(f"RMSE: {rmse:,.0f} SGD")
print(f"MAE: {mae:,.0f} SGD")
```

```
R²: 0.7962
Adjusted R²: 0.7961
RMSE: 93 SGD
MAE: 74 SGD
```

In [169…
```
coefs = model2.coef_
features = X_train_2.columns

# Create DataFrame for plotting
coef_df = pd.DataFrame({'Feature': features, 'Coefficient': coefs})
coef_df['Abs_Coefficient'] = np.abs(coef_df['Coefficient'])  # strength of impac
coef_df = coef_df.sort_values(by='Abs_Coefficient', ascending=False)

# Add direction and color
coef_df['Direction'] = coef_df['Coefficient'].apply(lambda x: 'Increase' if x >
coef_df['Color'] = coef_df['Direction'].map({'Increase': 'green', 'Decrease': 'r
```

```
# Sort by coefficient value (ascending: most negative to most positive)
coef_df = coef_df.sort_values(by='Coefficient')

# Plot
plt.figure(figsize=(12, 6))
bars = plt.barh(coef_df['Feature'], coef_df['Coefficient'], color=coef_df['Color
plt.axvline(0, color='black', linestyle='--', linewidth=0.8)
plt.xlabel("Effect on Resale Price (Standardized Coefficient)")
plt.title("Model 2 Coefficients: Linear Regression")

# Annotate each bar
for bar, coef in zip(bars, coef_df['Coefficient']):
    width = bar.get_width()
    label_x = width + (0.05 if coef > 0 else -0.05)
    ha = 'left' if coef > 0 else 'right'
    plt.text(label_x, bar.get_y() + bar.get_height() / 2,
             f"{coef:.2f}", va='center', ha=ha, color='black')

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



Model 2's residual plots demonstrates normality and homoscedasticity much better than model 1. Although model 2 is less interpretable as model 1.

Therefore model 1 should be used for feature strength interpretation while model 2 is tuned for prediction accuracy.

Model 3 -> Non-linear Decision Tree

In [16]:
```
geo_df_MRT_cbd = pd.read_csv('C:/Users/bulle/Desktop/HTX Interview/Kaggle_HDB/ge
```

```
C:\Users\bulle\AppData\Local\Temp\ipykernel_12920\74063598.py:1: DtypeWarning: Co
lumns (17) have mixed types. Specify dtype option on import or set low_memory=Fal
se.
  geo_df_MRT_cbd = pd.read_csv('C:/Users/bulle/Desktop/HTX Interview/Kaggle_HDB/g
eo_df_mrt_cbd.csv',index_col=0)
```

In [518…
```
df_model3= geo_df_MRT_cbd.copy()
```

In [520…

```
# 1: Calculate age of flat
current_year = 2020
df_model3['age_of_flat'] = current_year - df_model3['lease_commence_date']

# Convert 'storey_range' to numeric average
def parse_storey_range(s):
    try:
        low, high = map(int, s.split(' TO '))
        return (low + high) / 2
    except:
        return None

df_model3['storey_avg'] = df_model3['storey_range'].apply(parse_storey_range)

#3: Drop columns not used in modeling
df_model3.drop(columns=[
    'storey_range', 'block', 'street_name', 'flat_model', 'year',
    'latitude', 'longitude', 'x', 'y', 'postal', 'nearest_mrt',
    'lease_commence_date', 'flat_type', 'full_address'
], axis=1, inplace=True)

#4: Convert 'month' to numerical time indicator
df_model3['month'] = pd.to_datetime(df_model3['month'])
df_model3['months_since_start'] = (
    df_model3['month'].dt.to_period('M') - df_model3['month'].min().to_period('M
).apply(lambda x: x.n)
df_model3.drop(columns=['month'], axis=1, inplace=True)


#5: One-hot encode 'town'
df_model3 = pd.get_dummies(df_model3, columns=['town'], drop_first=True)
```

In [521…

```
df_model3
```

Out[521…

	floor_area_sqm	resale_price	nearest_mrt_dist_m	distance_to_cbd_m	age_of_flat
0	31.0	9000.0	826.278252	6834.444796	43
8279	148.0	160000.0	1181.168902	13929.656560	35
8280	151.0	150000.0	431.211489	13145.277170	35
8281	151.0	159600.0	431.211489	13145.277170	35
8282	149.0	193000.0	431.211489	13145.277170	35
...	...	...	...	...	..
762048	93.0	370000.0	1476.901513	12892.078360	7
762049	93.0	400888.0	1476.901513	12892.078360	7
762050	104.0	345000.0	1334.093222	13227.140710	27
762037	92.0	360000.0	1546.425808	12825.761210	5
766098	142.0	470000.0	676.296577	13897.311300	25

824732 rows × 33 columns

In [522…

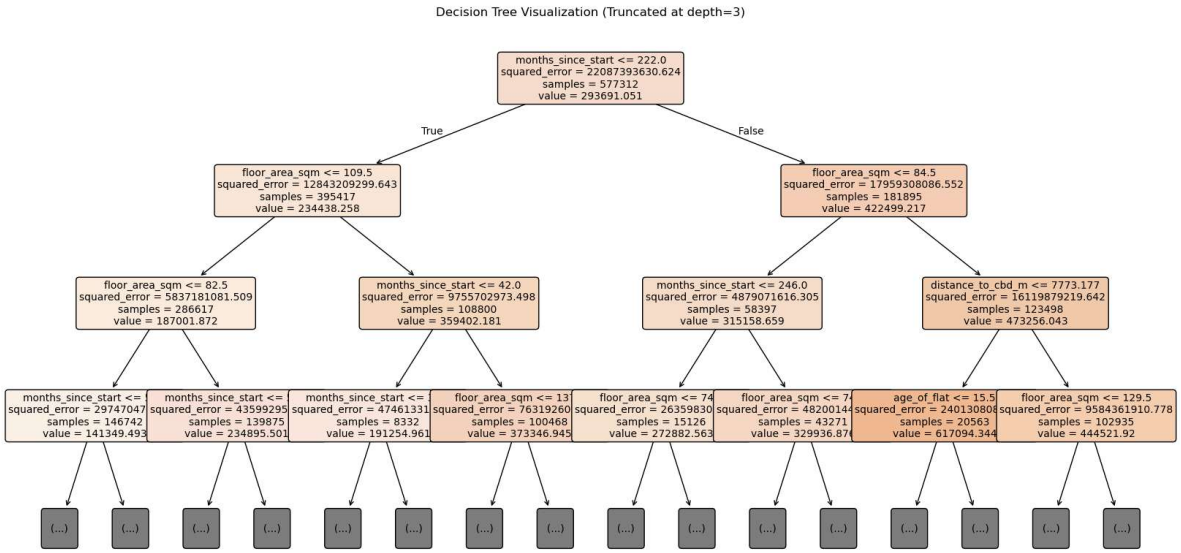
```
# Define features (X) and target (y)
X_3 = df_model3.drop(columns='resale_price')
y_3 = df_model3['resale_price']

# Train-test split
X_train_3, X_test_3, y_train_3, y_test_3 = train_test_split(
    X_3, y_3, test_size=0.3, random_state=101
)
```

In [523…

```
dt_model_3 = DecisionTreeRegressor(random_state=101, max_depth=5)  # You can tun
dt_model_3.fit(X_train_3, y_train_3)

# GridSearchCV for tuning
param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10]
}

grid_dt = GridSearchCV(DecisionTreeRegressor(random_state=101),
                       param_grid=param_grid,
                       cv=5,
                       scoring='neg_root_mean_squared_error',
                       n_jobs=-1,
                       verbose=1)
grid_dt.fit(X_train_3, y_train_3)

best_dt_model_3 = grid_dt.best_estimator_
print("✅ Best Params:", grid_dt.best_params_)

y_pred_3 = best_dt_model_3.predict(X_test_3)

mse = mean_squared_error(y_test_3, y_pred_3)
```

```
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_3, y_pred_3)
r2 = r2_score(y_test_3, y_pred_3)
adj_r2 = 1 - (1 - r2) * (len(y_test_3) - 1) / (len(y_test_3) - X_test_3.shape[1]

print(f"📊 R²: {r2:.4f}")
print(f"📊 Adjusted R²: {adj_r2:.4f}")
print(f"📊 RMSE: {rmse:,.0f} SGD")
print(f"📊 MAE: {mae:,.0f} SGD")

plt.figure(figsize=(20, 10))
plot_tree(best_dt_model_3,
          feature_names=X_3.columns,
          filled=True,
          rounded=True,
          fontsize=10,
          max_depth=3)  # Visualizing up to depth 3 for clarity
plt.title("Decision Tree Visualization (Truncated at depth=3)")
plt.show()
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
✅ Best Params: {'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split':
20}
📊 R²: 0.9709
📊 Adjusted R²: 0.9709
📊 RMSE: 25,428 SGD
📊 MAE: 17,612 SGD
```



In [534…

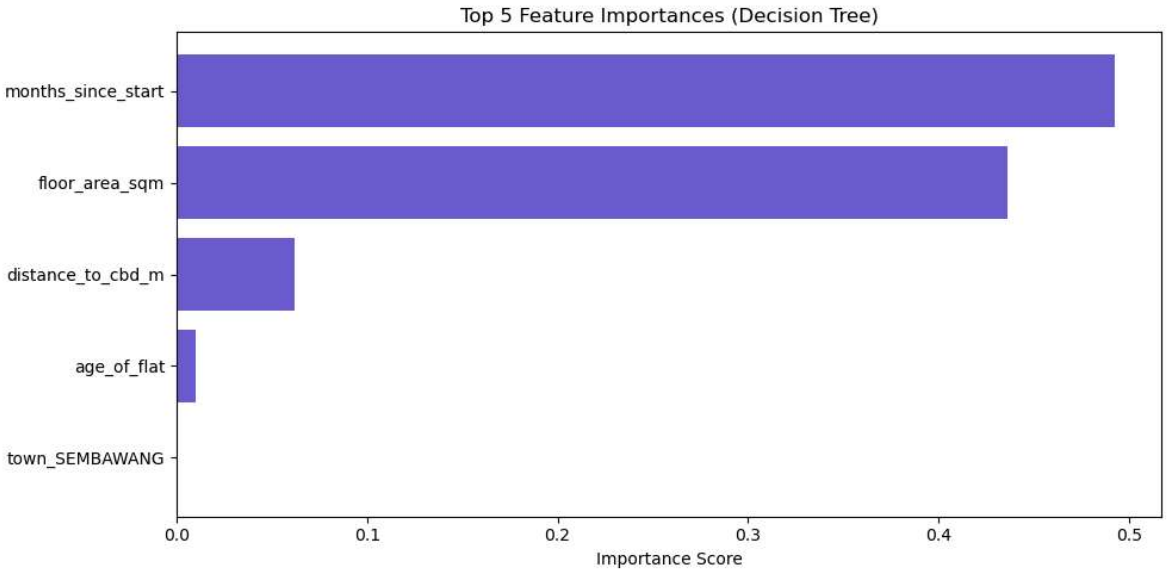
```
# Use the best model found by GridSearch
best_dt_model_3 = grid_dt.best_estimator_

# Cross-validate R²
cv_r2_scores = cross_val_score(best_dt_model_3, X_train_3, y_train_3, cv=5, scor
cv_r2_mean = np.mean(cv_r2_scores)
cv_r2_std = np.std(cv_r2_scores)

# Cross-validate RMSE (remember: scores are negative)
cv_rmse_scores = cross_val_score(best_dt_model_3, X_train_3, y_train_3, cv=5, sc
cv_rmse_mean = -np.mean(cv_rmse_scores)
cv_rmse_std = np.std(cv_rmse_scores)

# Print results
```

```
print(f"✅ CV R²: {cv_r2_mean:.4f} ± {cv_r2_std:.4f}")
print(f"✅ CV RMSE: {cv_rmse_mean:,.2f} ± {cv_rmse_std:,.2f} SGD")
```

```
✅ CV R²: 0.9691 ± 0.0002
✅ CV RMSE: 26,128.71 ± 28.54 SGD
```

In [524…

```
#plot top 5 important features

feat_importance = pd.DataFrame({
    'Feature': X_3.columns,
    'Importance': dt_model_3.feature_importances_
}).sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10,5))
plt.barh(feat_importance['Feature'][:5][::-1], feat_importance['Importance'][:5]
plt.title("Top 5 Feature Importances (Decision Tree)")
plt.xlabel("Importance Score")
plt.tight_layout()
plt.show()

#DT identifies "month_since_start" which is the year of transaction, and "floor_
#This is similar to the model 1's linear regression output
```



Model 4 -> Non-linear GradientBoosting(bin)

In [538…

```
#Use GradientBoosting (binned) instead of RF for faster performance
hgb_model_4 = HistGradientBoostingRegressor(random_state=42)
hgb_model_4.fit(X_train_3, y_train_3)

# --- Step 2: Predict on Test Set ---
y_pred_4 = hgb_model_4.predict(X_test_3)

# --- Step 3: Evaluate Model ---
mse_4 = mean_squared_error(y_test_3, y_pred_4)
rmse_4 = np.sqrt(mse_4)
mae_4 = mean_absolute_error(y_test_3, y_pred_4)
r2_4 = r2_score(y_test_3, y_pred_4)
adj_r2_4 = 1 - (1 - r2_4) * (len(y_test_3) - 1) / (len(y_test_3) - X_test_3.shap

# --- Step 4: Print Metrics ---
print(f"📊 R² (Gradient Boosting): {r2_4:.4f}")
```

```
print(f"📊 Adjusted R²: {adj_r2_4:.4f}")
print(f"📊 RMSE: {rmse_4:,.0f} SGD")
print(f"📊 MAE: {mae_4:,.0f} SGD")
```

```
📊 R² (Gradient Boosting): 0.9668
📊 Adjusted R²: 0.9668
📊 RMSE: 27,149 SGD
📊 MAE: 19,524 SGD
```

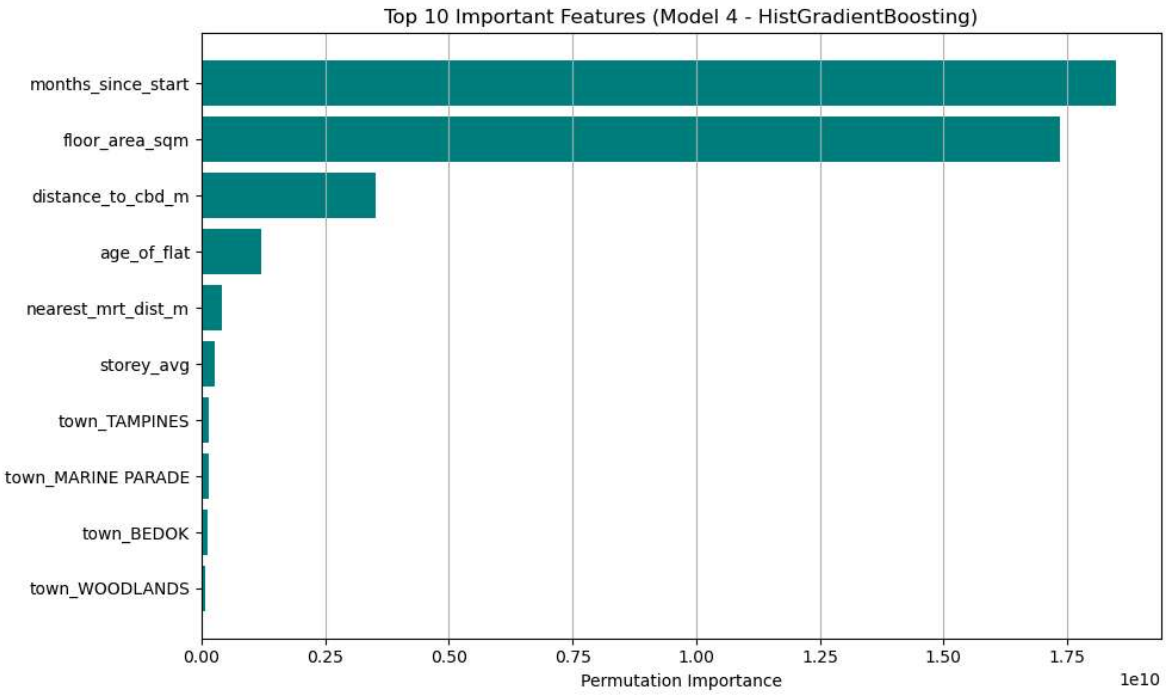
In [539…

```
# 5-fold cross-validation on training data
cv_r2_scores = cross_val_score(hgb_model_4, X_train_3, y_train_3, cv=5, scoring=
cv_r2_mean = np.mean(cv_r2_scores)
cv_r2_std = np.std(cv_r2_scores)

# Optionally, compute CV RMSE too
cv_rmse_scores = cross_val_score(hgb_model_4, X_train_3, y_train_3, cv=5, scorin
cv_rmse_mean = -np.mean(cv_rmse_scores)
cv_rmse_std = np.std(cv_rmse_scores)

# Print cross-validation results
print(f"✅ CV R²: {cv_r2_mean:.4f} ± {cv_r2_std:.4f}")
print(f"✅ CV RMSE: {cv_rmse_mean:,.2f} ± {cv_rmse_std:,.2f} SGD")
```

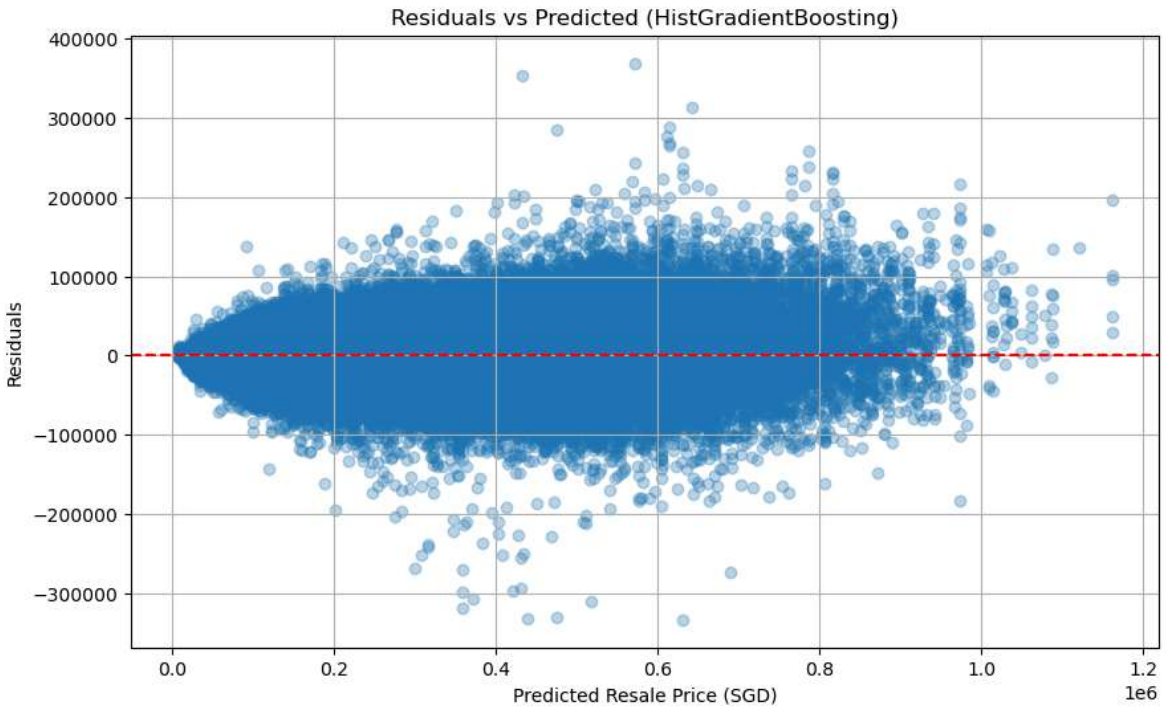
```
✅ CV R²: 0.9661 ± 0.0002
✅ CV RMSE: 27,379.21 ± 107.64 SGD
```

In [540…

```
#report permutation importance
result_4 = permutation_importance(
    hgb_model_4,
    X_test_3,
    y_test_3,
    n_repeats=10,
    random_state=42,
    scoring='neg_mean_squared_error'
)

# --- Step 2: Create DataFrame ---
importances_4 = pd.DataFrame({
    'Feature': X_3.columns,
    'Importance': result_4.importances_mean
}).sort_values(by='Importance', ascending=False)

# --- Step 3: Plot Top 10 Important Features ---
plt.figure(figsize=(10, 6))
plt.barh(importances_4['Feature'][:10][::-1], importances_4['Importance'][:10][:
plt.xlabel("Permutation Importance")
plt.title("Top 10 Important Features (Model 4 - HistGradientBoosting)")
plt.grid(True, axis='x')
plt.tight_layout()
plt.show()
```



In [67]:
```
import matplotlib.pyplot as plt

residuals = y_test_3 - y_pred_4
plt.figure(figsize=(10, 6))
plt.scatter(y_pred_3, residuals, alpha=0.3)
plt.axhline(0, linestyle='--', color='red')
plt.xlabel("Predicted Resale Price (SGD)")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted (HistGradientBoosting)")
plt.grid(True)
plt.show()
```



Insight: all four models consistently identify 'months_since_start' as a key variable influencing resale prices.

*** Interpretation *** Although proximity to the CBD and MRT stations is commonly perceived as a strong driver of resale flat prices in Singapore, our model results surprisingly suggests otherwise. Across historical data from 1990 to 2020, temporal factor 'months_since_start' and size of the HDB flats 'floor_area_sqm' exhibit substantially greater influence on resale price predictions. This implies that market-wide time effects (e.g., macroeconomic trends, policy changes, inflation) and flat size dominate in shaping prices, while locational factors such as distance_to_cbd_m play a comparatively secondary role in hdb resale prices. This may mean that location is not as significant in pushing housing prices as compared to time value and size of the flat.

As such, there is a need to investigate housing cooling measures, and black swan events and their impact on housing prices separately.

MODEL1 WITHOUT 'TOWN' VARIABLE AS IT WILL IMPLICITLY HAVE MULTI-COLLINEARITY WITH OUR "DISTANCE TO CBD" EFFECT. FOR APPLE TO APPLE COMPARISON WE REMOVE 'TOWN' FROM MODEL 1 FOR FAIR COMPARISON WITH MODEL 5

In [280…
```
df_model1_notown= df_model1.copy()
df_model1_notown.drop(columns=['town'], inplace=True)
df_model1_notown
```

Out[280…

	floor_area_sqm	resale_price	nearest_mrt_dist_m	distance_to_cbd_m	age_of_flat
0	31.0	9000.0	826.278252	6834.444796	43
1	148.0	160000.0	1181.168902	13929.656560	35
2	151.0	150000.0	431.211489	13145.277170	35
3	151.0	159600.0	431.211489	13145.277170	35
4	149.0	193000.0	431.211489	13145.277170	35
…	…	…	…	…	…
824727	93.0	370000.0	1476.901513	12892.078360	7
824728	93.0	400888.0	1476.901513	12892.078360	7
824729	104.0	345000.0	1334.093222	13227.140710	27
824730	92.0	360000.0	1546.425808	12825.761210	5
824731	142.0	470000.0	676.296577	13897.311300	25

824732 rows × 7 columns

In [286…
```
# ✅ Step 1: Split into X and y
X_notown = df_model1_notown.drop('resale_price', axis=1)
y_notown = df_model1_notown['resale_price']
X_train_notown, X_test_notown, y_train_notown, y_test_notown = train_test_split(

# ✅ Step 2: Standardize only numerical features
```

```
num_features_notown = ['floor_area_sqm', 'nearest_mrt_dist_m', 'distance_to_cbd_
                       'age_of_flat', 'storey_avg', 'months_since_start']

scaler = StandardScaler()
X_train_notown[num_features_notown] = scaler.fit_transform(X_train_notown[num_fe
X_test_notown[num_features_notown] = scaler.transform(X_test_notown[num_features
```

In [288…

```
model1_notown = LinearRegression()
model1_notown.fit(X_train_notown, y_train_notown)
```

Out[288…

LinearRegression

LinearRegression()

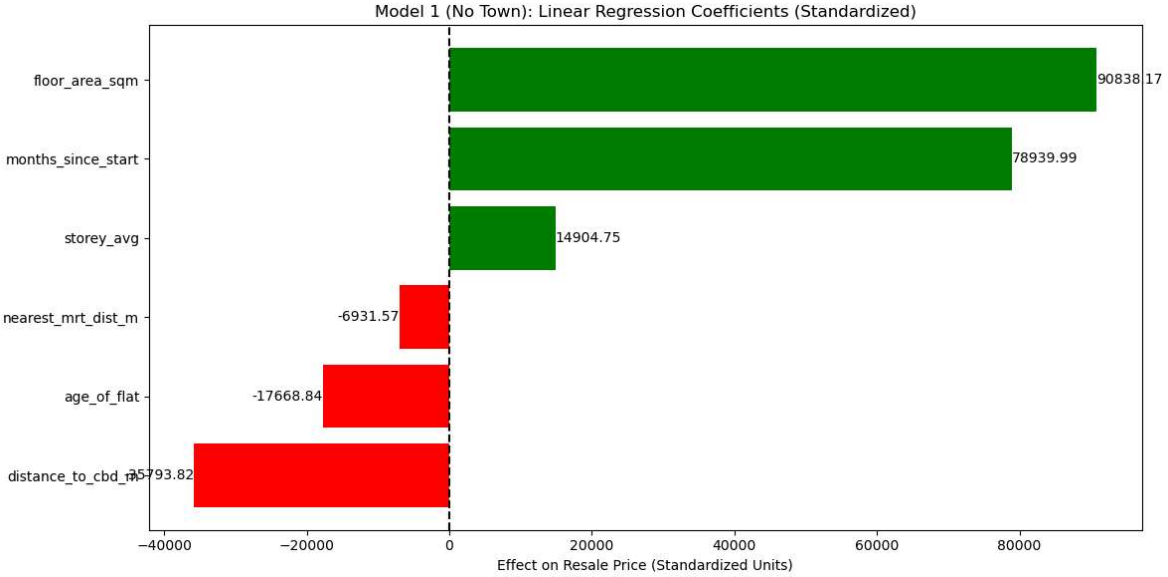
In [290…

```
# ✅ Step 4: Create DataFrame of coefficients
coef_df_notown = pd.DataFrame({
    'Feature': X_train_notown.columns,
    'Standardized Coef': model1_notown.coef_
})

# ✅ Step 5: Reverse standardization for numerical features
original_std_notown = scaler.scale_
coef_df_notown['Original Std'] = coef_df_notown['Feature'].apply(
    lambda f: original_std_notown[num_features_notown.index(f)] if f in num_feat
)
coef_df_notown['Per Unit Effect (SGD)'] = coef_df_notown['Standardized Coef'] /

# ✅ Step 6: Add readable units
units_notown = {
    'floor_area_sqm': 'sqm',
    'nearest_mrt_dist_m': 'meter from MRT',
    'distance_to_cbd_m': 'meter from CBD',
    'age_of_flat': 'year of age',
    'storey_avg': 'storey',
    'months_since_start': 'month (transaction timing)'
}
coef_df_notown['Unit'] = coef_df_notown['Feature'].map(units_notown)

# ✅ Step 7: Interpretation printout
print("🔍 Interpretation (Numerical Variables Only) – Model 1 without Town:\n")
for _, row in coef_df_notown.dropna(subset=['Original Std']).iterrows():
    effect = row['Per Unit Effect (SGD)']
    direction = "increases" if effect > 0 else "decreases"
    print(f"💡 For every additional 1 {row['Unit']}, resale price {direction} by

# ✅ Step 8: Plot standardized coefficients
coef_df_notown['Direction'] = coef_df_notown['Standardized Coef'].apply(lambda x
coef_df_notown['Color'] = coef_df_notown['Direction'].map({'Increase': 'green',
coef_df_notown = coef_df_notown.sort_values(by='Standardized Coef')

plt.figure(figsize=(12, 6))
bars = plt.barh(coef_df_notown['Feature'], coef_df_notown['Standardized Coef'],
plt.axvline(0, color='black', linestyle='--')
plt.xlabel("Effect on Resale Price (Standardized Units)")
plt.title("Model 1 (No Town): Linear Regression Coefficients (Standardized)")

for bar, coef in zip(bars, coef_df_notown['Standardized Coef']):
    width = bar.get_width()
```

```
    label_x = width + (0.05 if coef > 0 else -0.05)
    ha = 'left' if coef > 0 else 'right'
    plt.text(label_x, bar.get_y() + bar.get_height() / 2,
             f"{coef:.2f}", va='center', ha=ha, color='black')

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

🔍 Interpretation (Numerical Variables Only) – Model 1 without Town:

💡 For every additional 1 sqm, resale price increases by $3,489 SGD.
💡 For every additional 1 meter from MRT, resale price decreases by $18 SGD.
💡 For every additional 1 meter from CBD, resale price decreases by $9 SGD.
💡 For every additional 1 year of age, resale price decreases by $1,872 SGD.
💡 For every additional 1 storey, resale price increases by $3,224 SGD.
💡 For every additional 1 month (transaction timing), resale price increases by $831 SGD.



In [292…

```
# --- Evaluate Model 1 (No Town) ---
y_pred_notown = model1_notown.predict(X_test_notown)
r2_notown = r2_score(y_test_notown, y_pred_notown)
adj_r2_notown = 1 - (1 - r2_notown) * (len(y_test_notown) - 1) / (len(y_test_not
rmse_notown = np.sqrt(mean_squared_error(y_test_notown, y_pred_notown))
mae_notown = mean_absolute_error(y_test_notown, y_pred_notown)

print("📊 Model 1 (No Town) Evaluation:")
print(f"R²: {r2_notown:.4f}")
print(f"Adjusted R²: {adj_r2_notown:.4f}")
print(f"RMSE: ${rmse_notown:,.0f}")
print(f"MAE: ${mae_notown:,.0f}\n")
```

```
📊 Model 1 (No Town) Evaluation:
R²: 0.7721
Adjusted R²: 0.7721
RMSE: $71,124
MAE: $54,464
```

In [276…

```
df_model5= geo_df_MRT_cbd.copy()
df_model5
```

Out[276…

	month	town	flat_type	block	street_name	storey_range	floor_area_sqm
0	1990-01-01	ANG MO KIO	1 ROOM	309	ANG MO KIO AVE 1	10 TO 12	31.0
8279	1990-01-08	JURONG WEST	EXECUTIVE	558	JURONG WEST ST 42	10 TO 12	148.0
8280	1990-01-08	JURONG WEST	EXECUTIVE	472	JURONG WEST ST 41	10 TO 12	151.0
8281	1990-01-08	JURONG WEST	EXECUTIVE	472	JURONG WEST ST 41	10 TO 12	151.0
8282	1990-01-08	JURONG WEST	EXECUTIVE	472	JURONG WEST ST 41	10 TO 12	149.0
...	...	...	...	...	...	...	...
762048	2020-01-03	YISHUN	4 ROOM	458	YISHUN AVE 11	07 TO 09	93.0
762049	2020-01-03	YISHUN	4 ROOM	458	YISHUN AVE 11	10 TO 12	93.0
762050	2020-01-03	YISHUN	4 ROOM	416	YISHUN AVE 11	04 TO 06	104.0
762037	2020-01-03	YISHUN	4 ROOM	431D	YISHUN AVE 1	04 TO 06	92.0
766098	2020-01-07	CHOA CHU KANG	EXECUTIVE	526	CHOA CHU KANG ST 51	04 TO 06	142.0

824732 rows × 20 columns

In [190…

```
#0: select data from 2018 to 2020 only
df_model5['month'] = pd.to_datetime(df_model5['month'])
df_model5 = df_model5[(df_model5['month'] >= '2018-01-01') & (df_model5['month']

# 1: Calculate age of flat
current_year = 2020
df_model5['age_of_flat'] = current_year - df_model5['lease_commence_date']

# Convert 'storey_range' to numeric average
def parse_storey_range(s):
    try:
        low, high = map(int, s.split(' TO '))
        return (low + high) / 2
    except:
        return None
```

```
df_model5['storey_avg'] = df_model5['storey_range'].apply(parse_storey_range)

#3: Drop columns not used in modeling
df_model5.drop(columns=[
    'storey_range', 'block', 'street_name', 'flat_model', 'year',
    'latitude', 'longitude', 'x', 'y', 'postal', 'nearest_mrt',
    'lease_commence_date', 'flat_type', 'full_address','town'
], axis=1, inplace=True)

#4: Convert 'month' to numerical time indicator
df_model5['month'] = pd.to_datetime(df_model5['month'])
df_model5['months_since_start'] = (
    df_model5['month'].dt.to_period('M') - df_model5['month'].min().to_period('M
).apply(lambda x: x.n)
df_model5.drop(columns=['month'], axis=1, inplace=True)
```

In [216…

```
df_model5
```

Out[216…

	floor_area_sqm	resale_price	nearest_mrt_dist_m	distance_to_cbd_m	age_of_flat
718693	91.0	755000.0	344.682290	3762.934998	8
729072	122.0	400000.0	508.521495	15879.505840	23
727337	67.0	280000.0	722.915099	10888.052030	41
727336	67.0	275000.0	706.232032	11009.778990	41
727335	67.0	299000.0	472.363742	10880.761650	41
…	…	…	…	…	…
762048	93.0	370000.0	1476.901513	12892.078360	7
762049	93.0	400888.0	1476.901513	12892.078360	7
762050	104.0	345000.0	1334.093222	13227.140710	27
762037	92.0	360000.0	1546.425808	12825.761210	5
766098	142.0	470000.0	676.296577	13897.311300	25

59805 rows × 7 columns

In [218…
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Step 1: Split features and target
X_5 = df_model5.drop('resale_price', axis=1)
y_5 = df_model5['resale_price']

# Step 2: Train-test split
X_train_5, X_test_5, y_train_5, y_test_5 = train_test_split(X_5, y_5, test_size=

# Step 3: Standardize numerical features
num_features_5 = ['floor_area_sqm', 'nearest_mrt_dist_m', 'distance_to_cbd_m', '

scaler_5 = StandardScaler()
X_train_5[num_features_5] = scaler_5.fit_transform(X_train_5[num_features_5])
X_test_5[num_features_5] = scaler_5.transform(X_test_5[num_features_5])
```

In [220…
```
num_features_5
```

Out[220…
```
['floor_area_sqm',
 'nearest_mrt_dist_m',
 'distance_to_cbd_m',
 'age_of_flat',
 'storey_avg',
 'months_since_start']
```

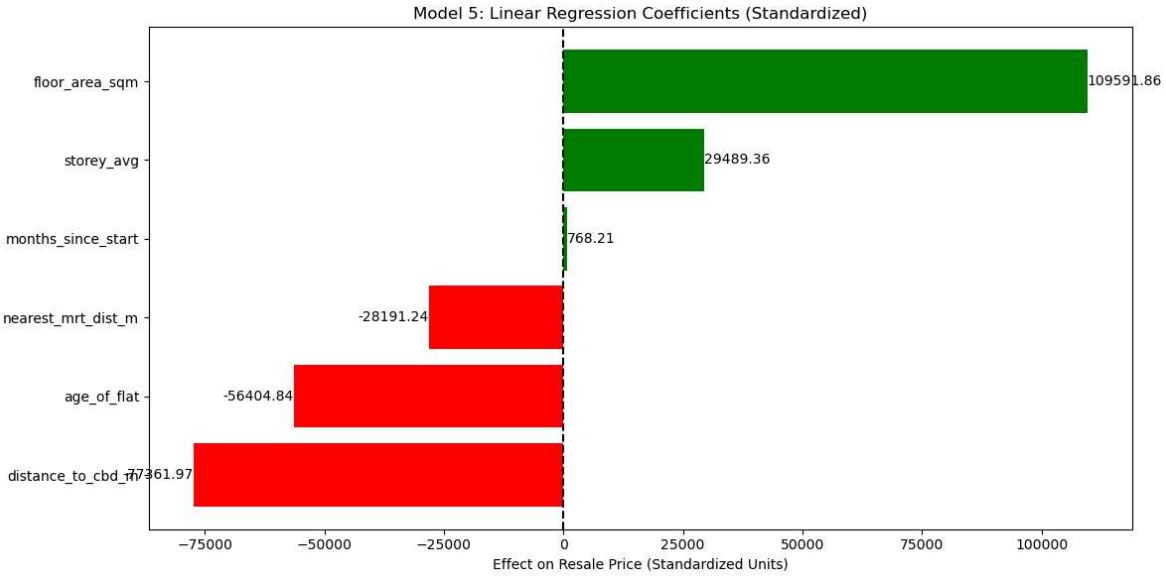
In [222…
```
model5_LR = LinearRegression()
model5_LR.fit(X_train_5, y_train_5)
```

Out[222…
```
▾ LinearRegression
LinearRegression()
```

In [231…
```
coef_df_5 = pd.DataFrame({
    'Feature': X_train_5.columns,
    'Standardized Coef': model5_LR.coef_
})

# --- Step 3: Reverse standardization only for numerical features
num_features_5 = ['floor_area_sqm', 'nearest_mrt_dist_m', 'distance_to_cbd_m',
                  'age_of_flat', 'storey_avg','months_since_start']

# Store std values from the scaler
original_std_5 = scaler_5.scale_

# Add std values into the DataFrame
coef_df_5['Original Std'] = coef_df_5['Feature'].apply(
    lambda f: original_std_5[num_features_5.index(f)] if f in num_features_5 els
)

# Compute original scale coefficients
coef_df_5['Per Unit Effect (SGD)'] = coef_df_5['Standardized Coef'] / coef_df_5[

# Add unit description
units_5 = {
    'floor_area_sqm': 'sqm',
    'age_of_flat': 'year of age',
    'storey_avg': 'storey',
```

```
    'nearest_mrt_dist_m': 'meter from MRT',
    'distance_to_cbd_m': 'meter from CBD',
    'months_since_start': 'month (transaction timing)'
}
coef_df_5['Unit'] = coef_df_5['Feature'].map(units_5)

# --- Step 4: Interpret only numerical features
print("🔍 Interpretation (Numerical Variables Only):\n")
for _, row in coef_df_5.dropna().iterrows():
    effect = row['Per Unit Effect (SGD)']
    direction = "increases" if effect > 0 else "decreases"
    print(f"💡 For every additional 1 {row['Unit']}, resale price {direction} by

# --- Step 5: Plot all coefficients (standardized)
coef_df_5['Direction'] = coef_df_5['Standardized Coef'].apply(lambda x: 'Increas
coef_df_5['Color'] = coef_df_5['Direction'].map({'Increase': 'green', 'Decrease'
coef_df_5 = coef_df_5.sort_values(by='Standardized Coef')

plt.figure(figsize=(12, 6))
bars = plt.barh(coef_df_5['Feature'], coef_df_5['Standardized Coef'], color=coef
plt.axvline(0, color='black', linestyle='--')
plt.xlabel("Effect on Resale Price (Standardized Units)")
plt.title("Model 5: Linear Regression Coefficients (Standardized)")

for bar, coef in zip(bars, coef_df_5['Standardized Coef']):
    width = bar.get_width()
    label_x = width + (0.05 if coef > 0 else -0.05)
    ha = 'left' if coef > 0 else 'right'
    plt.text(label_x, bar.get_y() + bar.get_height() / 2,
             f"{coef:.2f}", va='center', ha=ha, color='black')

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

```
🔍 Interpretation (Numerical Variables Only):

💡 For every additional 1 sqm, resale price increases by $4,486 SGD.
💡 For every additional 1 meter from MRT, resale price decreases by $62 SGD.
💡 For every additional 1 meter from CBD, resale price decreases by $20 SGD.
💡 For every additional 1 year of age, resale price decreases by $4,287 SGD.
💡 For every additional 1 storey, resale price increases by $5,126 SGD.
💡 For every additional 1 month (transaction timing), resale price increases by
$81 SGD.
```



In [264…
```
# --- Evaluate Model 5 ---
y_pred_5 = model5_LR.predict(X_test_5)
r2_5 = r2_score(y_test_5, y_pred_5)
adj_r2_5 = 1 - (1 - r2_5) * (len(y_test_5) - 1) / (len(y_test_5) - X_test_5.shap
rmse_5 = np.sqrt(mean_squared_error(y_test_5, y_pred_5))
mae_5 = mean_absolute_error(y_test_5, y_pred_5)

print("📊 Model 5 (2018–2020, No Town) Evaluation:")
print(f"R²: {r2_5:.4f}")
print(f"Adjusted R²: {adj_r2_5:.4f}")
print(f"RMSE: ${rmse_5:,.0f}")
print(f"MAE: ${mae_5:,.0f}")
```

```
📊 Model 5 (2018–2020, No Town) Evaluation:
R²: 0.7961
Adjusted R²: 0.7960
RMSE: $69,741
MAE: $53,859
```

INSIGHT: This is a critical observation. By selecting a smaller timeframe of data from 2018 to 2020, we can see the effects of time value "months_since_start" diminishing. While the effects of "distance to cbd" and "nearest mrt dist" amplified. This means that in the short time frame, such effects are more important than in the longer time frame.

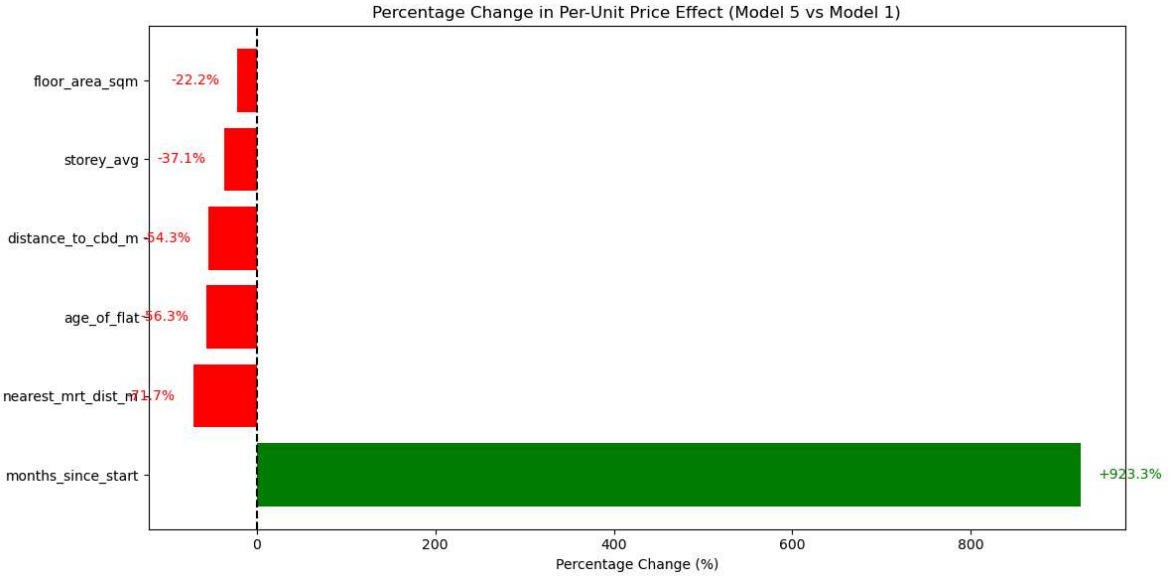
-> The effects of the different variables have significantly different effects when we are analysing in the shorter term vs longer term period!

-> Model 1 (without town variable) vs Model 5 (without town variable) demonstrates that inflation has a compounding effect over longer time frames that affected our housing prices strongly in the long-run.

(note: this time frame effect is worth investigating)

In [347…
```
import matplotlib.pyplot as plt
import numpy as np

# Merge both models into one dataframe for comparison
comparison_df = pd.merge(
    coef_df_notown[['Feature', 'Per Unit Effect (SGD)']].rename(columns={'Per Un
    coef_df_5[['Feature', 'Per Unit Effect (SGD)']].rename(columns={'Per Unit Ef
    on='Feature'
)

# Sort by average effect
comparison_df['avg_effect'] = comparison_df[['Model 1', 'Model 5']].mean(axis=1)
comparison_df = comparison_df.sort_values(by='avg_effect')

# Add % change column
comparison_df['% Change'] = 100 * (comparison_df['Model 1'] - comparison_df['Mod

# Sort by absolute % change for emphasis
comparison_df_sorted = comparison_df.sort_values('% Change', key=abs, ascending=

# Set bar colors based on sign
```

```
colors = ['green' if x > 0 else 'red' for x in comparison_df_sorted['% Change']]

# Plot
plt.figure(figsize=(12, 6))
bars = plt.barh(
    comparison_df_sorted['Feature'],
    comparison_df_sorted['% Change'],
    color=colors
)
plt.axvline(0, color='black', linestyle='--')
plt.title('Percentage Change in Per-Unit Price Effect (Model 5 vs Model 1)')
plt.xlabel('Percentage Change (%)')

# Annotate each bar
for bar, pct in zip(bars, comparison_df_sorted['% Change']):
    width = bar.get_width()
    label_x = width + (20 if width > 0 else -20)
    ha = 'left' if width > 0 else 'right'
    plt.text(label_x, bar.get_y() + bar.get_height() / 2, f"{pct:+.1f}%", ha=ha,

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



In [339...

```
comparison_df
```

Out[339...

	Feature	Model 1	Model 5	avg_effect	% Change
1	age_of_flat	-1871.729744	-4286.747682	-3079.238713	-56.336834
2	nearest_mrt_dist_m	-17.552398	-62.080145	-39.816272	-71.726229
0	distance_to_cbd_m	-9.056666	-19.813399	-14.435032	-54.290196
4	months_since_start	830.644252	81.175992	455.910122	923.263443
5	floor_area_sqm	3488.906186	4486.302270	3987.604228	-22.232030
3	storey_avg	3224.428095	5126.228009	4175.328052	-37.099402

CONCLUSION:

📈 months_since_start's (Transaction timing) effect exploded over a longer time period: The per-unit **impact of transaction timing increased dramatically by +923.3%** in Model 1 (1990-2020) compared to Model 5 (2018–2020) i.e. Transaction timing is the key HDB resale price driver over the long term, likely due to macro factors like inflation, market cycles, and policy shifts. This also indicates that the macro economics have a much stronger compounding effect on HDB resale prices than just proximity and distance effect to MRT and CBD.

📉 nearest_mrt_dist_m (Distance to nearest MRT), age_of_flat, and distance_to_cbd_m (Distance to Orchard MRT/ CBD) effect diminished over a longer time frame: Their effects was much less negative in Model 1. This suggests location-related features and age of flat had a stronger influence in recent years (Model 5), possibly due to more buyer sensitivity to location and age of flat during stable market phases. But, the increase in HDB resale prices

Among all predictors, floor area and storey level consistently demonstrate a strong, positive correlation with resale price across both long-term and short-term models. Their influence appears stable and significant, making them robust baseline drivers of housing value

In [ ]: