AirBNB_Quadratic

September 7, 2025

[1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
# Load the dataset
df=pd.read_csv(r'D:\Quadratic\cleaned_airbnb_data2.csv')
```

[2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52966 entries, 0 to 52965
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   room_type             52966 non-null  object
 1   accommodates          52966 non-null  int64
 2   bathrooms             52966 non-null  float64
 3   cancellation_policy   52966 non-null  object
 4   cleaning_fee          52966 non-null  int64
 5   instant_bookable      52966 non-null  object
 6   review_scores_rating  52966 non-null  int64
 7   bedrooms              52966 non-null  int64
 8   beds                  52966 non-null  int64
 9   log_price             52966 non-null  float64
 10  actual_price          52966 non-null  float64
dtypes: float64(3), int64(5), object(3)
memory usage: 4.4+ MB
```

[3]:
```
df.isnull().sum()
```

[3]:
```
room_type               0
accommodates            0
bathrooms               0
```

```
cancellation_policy     0
cleaning_fee            0
instant_bookable        0
review_scores_rating    0
bedrooms                0
beds                    0
log_price               0
actual_price            0
dtype: int64
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52966 entries, 0 to 52965
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   room_type             52966 non-null  object
 1   accommodates          52966 non-null  int64
 2   bathrooms             52966 non-null  float64
 3   cancellation_policy   52966 non-null  object
 4   cleaning_fee          52966 non-null  int64
 5   instant_bookable      52966 non-null  object
 6   review_scores_rating  52966 non-null  int64
 7   bedrooms              52966 non-null  int64
 8   beds                  52966 non-null  int64
 9   log_price             52966 non-null  float64
 10  actual_price          52966 non-null  float64
dtypes: float64(3), int64(5), object(3)
memory usage: 4.4+ MB
```

[5]:
```
df['actual_price'] = np.exp(df['log_price']) - 1
```

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52966 entries, 0 to 52965
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   room_type             52966 non-null  object
 1   accommodates          52966 non-null  int64
 2   bathrooms             52966 non-null  float64
 3   cancellation_policy   52966 non-null  object
 4   cleaning_fee          52966 non-null  int64
 5   instant_bookable      52966 non-null  object
 6   review_scores_rating  52966 non-null  int64
 7   bedrooms              52966 non-null  int64
 8   beds                  52966 non-null  int64
 9   log_price             52966 non-null  float64
 10  actual_price          52966 non-null  float64
dtypes: float64(3), int64(5), object(3)
memory usage: 4.4+ MB
```

[7]:
```
df.describe()
```

[7]:
```
       accommodates     bathrooms  cleaning_fee  review_scores_rating  \
count  52966.000000  52966.000000  52966.000000          52966.000000
mean       3.551712      1.309066      0.745459             93.681758
std        2.356838      0.655386      0.435607              7.700825
min        1.000000      0.000000      0.000000             20.000000
25%        2.000000      1.000000      0.000000             92.000000
50%        3.000000      1.000000      1.000000             96.000000
75%        4.000000      1.500000      1.000000             98.000000
max       16.000000      8.000000      1.000000            100.000000

           bedrooms          beds     log_price  actual_price
count  52966.000000  52966.000000  52966.000000  52966.000000
mean       1.370105      1.930446      4.871222    178.668316
std        0.957452      1.392457      0.758759    189.521051
min        0.000000      0.000000      0.000000      0.000000
25%        1.000000      1.000000      4.369448     78.000000
50%        1.000000      1.000000      4.828314    124.000000
75%        2.000000      2.000000      5.298317    199.000000
max       10.000000     18.000000      7.600402   1998.000001
```

[8]:
```
df.head(10)
```

[8]:
```
         room_type  accommodates  bathrooms cancellation_policy  cleaning_fee  \
0  Entire home/apt             3        1.0              strict             1
1  Entire home/apt             7        1.0              strict             1
2  Entire home/apt             5        1.0            moderate             1
3  Entire home/apt             4        1.0            flexible             1
4  Entire home/apt             2        1.0            moderate             1
5     Private room             2        1.0              strict             1
6  Entire home/apt             3        1.0            moderate             1
7  Entire home/apt             2        1.0            moderate             1
8     Private room             2        1.0            moderate             1
9     Private room             2        1.0            moderate             1

  instant_bookable  review_scores_rating  bedrooms  beds  log_price  \
0                f                   100         1     1   5.010635
1                t                    93         3     3   5.129899
2                t                    92         1     3   4.976734
3                f                    96         2     2   6.620073
4                t                    40         0     1   4.744932
```

```
5                t                   100          1      1   4.442651
6                t                    97          1      1   4.418841
7                f                    93          1      1   4.787492
8                f                    99          1      1   4.787492
9                t                    90          1      1   3.583519

   actual_price
0         149.0
1         168.0
2         144.0
3         749.0
4         114.0
5          84.0
6          82.0
7         119.0
8         119.0
9          35.0
```

1 Avg no of Bathrooms VS Room Type

[9]:
```python
import pandas as pd
import plotly.express as px

# Convert 'bathrooms' column to numeric (handling errors)
df['bathrooms'] = pd.to_numeric(df['bathrooms'], errors='coerce')

# Group by 'room_type' and calculate the average number of bathrooms
room_bath_avg = df.groupby('room_type')['bathrooms'].mean().reset_index()
room_bath_avg['bathrooms'] = room_bath_avg['bathrooms'].round(2)

# Count occurrences of each room type
room_counts = df['room_type'].value_counts().reset_index()
room_counts.columns = ['room_type', 'count']

# Merge average bathrooms with room counts
room_summary = pd.merge(room_bath_avg, room_counts, on='room_type')

# Create a bar chart
fig = px.bar(
    room_summary,
    x='room_type',
    y='bathrooms',
    color='room_type',
    text='bathrooms',
    title='Average Number of Bathrooms by Room Type',
    labels={'bathrooms': 'Average Bathrooms', 'room_type': 'Room Type'},
    height=500
)

# Add count information as annotations
for i, row in room_summary.iterrows():
    fig.add_annotation(
        x=row['room_type'],
        y=row['bathrooms'] + 0.1,  # Adjust position for count annotation
        text=f"n={row['count']}",
        showarrow=False
    )

# Update layout for aesthetics
fig.update_layout(
    xaxis=dict(title='Room Type'),
    yaxis=dict(title='Average Number of Bathrooms'),
    plot_bgcolor='white'
)

# Display the plot
# fig.show()
```

2 Avg no of Accommodates VS Room Type

[10]:
```
# Convert 'accommodates' column to numeric (handling errors)
df['accommodates'] = pd.to_numeric(df['accommodates'], errors='coerce')

# Group by 'room_type' and calculate the average number of accommodates
room_acc_avg = df.groupby('room_type')['accommodates'].mean().reset_index()
room_acc_avg['accommodates'] = room_acc_avg['accommodates'].round(2)

# Count occurrences of each room type
room_counts = df['room_type'].value_counts().reset_index()
room_counts.columns = ['room_type', 'count']

# Merge average accommodates with room counts
room_summary = pd.merge(room_acc_avg, room_counts, on='room_type')

# Create a bar chart
fig = px.bar(
    room_summary,
    x='room_type',
    y='accommodates',
    color='room_type',
    text='accommodates',
    title='Average Number of Accommodates by Room Type',
    labels={'accommodates': 'Average Accommodates', 'room_type': 'Room Type'},
    height=500
)

# Add count information as annotations
for i, row in room_summary.iterrows():
    fig.add_annotation(
        x=row['room_type'],
        y=row['accommodates'] + 0.2,  # Adjust position for better visibility
        text=f"n={row['count']}",
        showarrow=False
    )

# Update layout
fig.update_layout(
    xaxis=dict(title='Room Type'),
    yaxis=dict(title='Average Number of Accommodates'),
    plot_bgcolor='white'
)

# Display the plot
# fig.show()
```

3 Relationship B/W Accommodates and Log Price

[11]:
```
# Convert relevant columns to numeric (handling errors)
df['accommodates'] = pd.to_numeric(df['accommodates'], errors='coerce')
df['log_price'] = pd.to_numeric(df['log_price'], errors='coerce')

# Remove rows with missing values in 'accommodates' and 'log_price'
df = df.dropna(subset=['accommodates', 'log_price'])

# Create a scatter plot with trend line
fig = px.scatter(
    df,
    x='accommodates',
    y='log_price',
    color='room_type',
    title='Relationship Between Accommodates and Log Price',
    labels={'accommodates': 'Number of People Accommodated', 'log_price': 'Log␣
 ↪Price'},
    trendline='ols',  # Ordinary Least Squares Regression
    height=500
)

# Update layout
```

```
fig.update_layout(
    xaxis=dict(title='Number of People Accommodated'),
    yaxis=dict(title='Log Price'),
    plot_bgcolor='white'
)

# Add an annotation for better explanation
fig.add_annotation(
    x=0.5,
    y=0.05,
    xref="paper",
    yref="paper",
    text="Each point represents a listing. The trend line shows the overall␣
 ↪relationship.",
    showarrow=False,
    font=dict(size=10)
)

# Show the figure
# fig.show()
```

4 Avg no of Bedrooms VS Room Type

[12]:
```
# Convert 'bedrooms' column to numeric (handling errors)
df['bedrooms'] = pd.to_numeric(df['bedrooms'], errors='coerce')

# Group by 'room_type' and calculate the average number of bedrooms
room_bed_avg = df.groupby('room_type')['bedrooms'].mean().reset_index()
room_bed_avg['bedrooms'] = room_bed_avg['bedrooms'].round(2)

# Count occurrences of each room type
room_counts = df['room_type'].value_counts().reset_index()
room_counts.columns = ['room_type', 'count']

# Merge average bedrooms with room counts
room_summary = pd.merge(room_bed_avg, room_counts, on='room_type')

# Create a bar chart
fig = px.bar(
    room_summary,
    x='room_type',
    y='bedrooms',
    color='room_type',
    text='bedrooms',
    title='Average Number of Bedrooms by Room Type',
    labels={'bedrooms': 'Average Bedrooms', 'room_type': 'Room Type'},
    height=500
)

# Add count information as annotations
for i, row in room_summary.iterrows():
    fig.add_annotation(
        x=row['room_type'],
        y=row['bedrooms'] + 0.1,  # Adjust position for better visibility
        text=f"n={row['count']}",
        showarrow=False
    )

# Update layout
fig.update_layout(
    xaxis=dict(title='Room Type'),
    yaxis=dict(title='Average Number of Bedrooms'),
    plot_bgcolor='white'
)

# Display the plot
# fig.show()
```

5 Avg Log Price VS Cancellation Policy

[13]:
```
# Convert 'log_price' column to numeric (handling errors)
df['log_price'] = pd.to_numeric(df['log_price'], errors='coerce')

# Group by 'cancellation_policy' and calculate the average log_price
policy_price_avg = df.groupby('cancellation_policy')['log_price'].mean().
 ↪reset_index()
policy_price_avg['log_price'] = policy_price_avg['log_price'].round(3)

# Count occurrences of each cancellation policy
policy_counts = df['cancellation_policy'].value_counts().reset_index()
policy_counts.columns = ['cancellation_policy', 'count']

# Merge average log_price with policy counts
policy_summary = pd.merge(policy_price_avg, policy_counts,␣
 ↪on='cancellation_policy')

# Sort by average log_price
policy_summary = policy_summary.sort_values('log_price')

# Create a bar chart
fig = px.bar(
    policy_summary,
    x='cancellation_policy',
    y='log_price',
    color='cancellation_policy',
    text='log_price',
    title='Average Log Price by Cancellation Policy',
    labels={'log_price': 'Average Log Price', 'cancellation_policy':␣
↪'Cancellation Policy'},
    height=500
)

# Add count information as annotations
for i, row in policy_summary.iterrows():
    fig.add_annotation(
        x=row['cancellation_policy'],
        y=row['log_price'] + 0.05,  # Adjust position for better visibility
        text=f"n={row['count']}",
        showarrow=False
    )

# Update layout
fig.update_layout(
    xaxis=dict(title='Cancellation Policy'),
    yaxis=dict(title='Average Log Price'),
    plot_bgcolor='white'
)

# Display the plot
# fig.show()
```

6 Avg Log Price VS Accommodates

[14]:
```
# Convert relevant columns to numeric (handling errors)
df['accommodates'] = pd.to_numeric(df['accommodates'], errors='coerce')
df['log_price'] = pd.to_numeric(df['log_price'], errors='coerce')

# Remove rows with missing values in 'accommodates' and 'log_price'
df = df.dropna(subset=['accommodates', 'log_price'])

# Group by accommodates and calculate average log_price and count
acc_price = df.groupby('accommodates')['log_price'].agg(['mean', 'count']).
↪reset_index()
acc_price['mean'] = acc_price['mean'].round(3)

# Create a bar chart
fig = px.bar(
    acc_price,
    x='accommodates',
    y='mean',
    text='mean',
    title='Average Log Price by Number of People Accommodated',
    labels={'accommodates': 'Number of People Accommodated', 'mean': 'Average␣
 ↪Log Price'},
    height=500,
    color='accommodates'
)

# Add count information as annotations
for i, row in acc_price.iterrows():
    fig.add_annotation(
        x=row['accommodates'],
        y=row['mean'] + 0.1,  # Adjust position for better visibility
        text=f"n={row['count']}",
        showarrow=False
    )

# Add a trend line
x_trend = acc_price['accommodates']
y_trend = acc_price['mean']
fig.add_trace(go.Scatter(
    x=x_trend,
    y=y_trend,
    mode='lines',
    name='Trend',
    line=dict(color='black', dash='dash')
))

# Update layout
fig.update_layout(
    xaxis=dict(title='Number of People Accommodated', tickmode='linear'),
    yaxis=dict(title='Average Log Price'),
    plot_bgcolor='white'
)

# Display the plot
# fig.show()
```

7 Dist of Log Price VS No of Accommodates

[15]:
```
# Convert relevant columns to numeric (handling errors)
df['accommodates'] = pd.to_numeric(df['accommodates'], errors='coerce')
df['log_price'] = pd.to_numeric(df['log_price'], errors='coerce')
```

```
# Remove rows with missing values in 'accommodates' and 'log_price'
df = df.dropna(subset=['accommodates', 'log_price'])

# Create a box plot to show distribution of log_price by accommodates
fig = px.box(
    df,
    x='accommodates',
    y='log_price',
    title='Distribution of Log Price by Number of People Accommodated',
    labels={'accommodates': 'Number of People Accommodated', 'log_price': 'Log␣
↪Price'},
    height=500,
    color='accommodates'
)

# Add a trend line using mean values
acc_price_mean = df.groupby('accommodates')['log_price'].mean().reset_index()
fig.add_trace(go.Scatter(
    x=acc_price_mean['accommodates'],
    y=acc_price_mean['log_price'],
    mode='lines+markers',
    name='Mean Log Price',
    line=dict(color='black', width=2),
    marker=dict(size=8, color='black')
))

# Add count annotations
acc_counts = df.groupby('accommodates').size().reset_index(name='count')
for i, row in acc_counts.iterrows():
    if row['count'] > 100:  # Only show for categories with significant counts
        fig.add_annotation(
            x=row['accommodates'],
            y=df[df['accommodates'] == row['accommodates']]['log_price'].max() +␣
↪0.1,
            text=f"n={row['count']}",
            showarrow=False
        )

# Update layout
fig.update_layout(
    xaxis=dict(title='Number of People Accommodated', tickmode='linear'),
    yaxis=dict(title='Log Price'),
    plot_bgcolor='white',
    boxmode='group'
)

# Display the plot
# fig.show()
```

8 Avg Log Price VS No of Bedrooms

[16]:
```
# Convert relevant columns to numeric (handling errors)
df['bedrooms'] = pd.to_numeric(df['bedrooms'], errors='coerce')
df['log_price'] = pd.to_numeric(df['log_price'], errors='coerce')

# Remove rows with missing values in 'bedrooms' and 'log_price'
df = df.dropna(subset=['bedrooms', 'log_price'])

# Group by bedrooms and calculate average log_price and count
bed_price = df.groupby('bedrooms')['log_price'].agg(['mean', 'count']).
 ↪reset_index()
bed_price['mean'] = bed_price['mean'].round(3)

# Create a bar chart
fig = px.bar(
    bed_price,
    x='bedrooms',
    y='mean',
    text='mean',
    title='Average Log Price by Number of Bedrooms',
    labels={'bedrooms': 'Number of Bedrooms', 'mean': 'Average Log Price'},
    height=500,
    color='bedrooms'
)

# Add count information as annotations
for i, row in bed_price.iterrows():
    fig.add_annotation(
        x=row['bedrooms'],
        y=row['mean'] + 0.1,  # Adjust position for better visibility
        text=f"n={row['count']}",
        showarrow=False
    )

# Add a trend line
x_trend = bed_price['bedrooms']
y_trend = bed_price['mean']
fig.add_trace(go.Scatter(
    x=x_trend,
    y=y_trend,
    mode='lines',
    name='Trend',
    line=dict(color='black', dash='dash')
))

# Update layout
fig.update_layout(
    xaxis=dict(title='Number of Bedrooms', tickmode='linear'),
    yaxis=dict(title='Average Log Price'),
    plot_bgcolor='white'
)

# Display the plot
# fig.show()
```

9 Avg Log Price VS No of Bathrooms

[17]:
```
# Convert relevant columns to numeric (handling errors)
df['bathrooms'] = pd.to_numeric(df['bathrooms'], errors='coerce')
df['log_price'] = pd.to_numeric(df['log_price'], errors='coerce')

# Remove rows with missing values in 'bathrooms' and 'log_price'
df = df.dropna(subset=['bathrooms', 'log_price'])

# Group by bathrooms and calculate average log_price and count
bath_price = df.groupby('bathrooms')['log_price'].agg(['mean', 'count']).
 ↪reset_index()
bath_price['mean'] = bath_price['mean'].round(3)

# Create a bar chart
fig = px.bar(
    bath_price,
    x='bathrooms',
    y='mean',
    text='mean',
    title='Average Log Price by Number of Bathrooms',
    labels={'bathrooms': 'Number of Bathrooms', 'mean': 'Average Log Price'},
    height=500,
    color='bathrooms'
)

# Add count information as annotations
for i, row in bath_price.iterrows():
    fig.add_annotation(
        x=row['bathrooms'],
        y=row['mean'] + 0.1,  # Adjust position for better visibility
        text=f"n={row['count']}",
        showarrow=False
    )
```

```
# Add a trend line
x_trend = bath_price['bathrooms']
y_trend = bath_price['mean']
fig.add_trace(go.Scatter(
    x=x_trend,
    y=y_trend,
    mode='lines',
    name='Trend',
    line=dict(color='black', dash='dash')
))

# Update layout
fig.update_layout(
    xaxis=dict(title='Number of Bathrooms', tickmode='linear'),
    yaxis=dict(title='Average Log Price'),
    plot_bgcolor='white'
)

# Display the plot
# fig.show()
```

10 Log Price VS Room Type

[18]:
```
# Convert 'log_price' to numeric (handling errors)
df['log_price'] = pd.to_numeric(df['log_price'], errors='coerce')

# Remove rows with missing values in 'room_type' and 'log_price'
df = df.dropna(subset=['room_type', 'log_price'])

# Group by 'room_type' and calculate average log_price and count
room_price_avg = df.groupby('room_type')['log_price'].agg(['mean', 'count']).
 ↪reset_index()
room_price_avg['mean'] = room_price_avg['mean'].round(3)

# Create a bar chart
fig = px.bar(
    room_price_avg,
    x='room_type',
    y='mean',
    color='room_type',
    text='mean',
    title='Average Log Price by Room Type',
    labels={'mean': 'Average Log Price', 'room_type': 'Room Type'},
    height=500
)
```

```python
# Add count information as annotations
for i, row in room_price_avg.iterrows():
    fig.add_annotation(
        x=row['room_type'],
        y=row['mean'] + 0.1,  # Adjust position for better visibility
        text=f"n={row['count']}",
        showarrow=False
    )

# Create a box plot to show distribution of log_price by room_type
fig2 = px.box(
    df,
    x='room_type',
    y='log_price',
    color='room_type',
    title='Distribution of Log Price by Room Type',
    labels={'log_price': 'Log Price', 'room_type': 'Room Type'},
    height=500
)

# Combine the two visualizations (Bar chart and Box plot)
fig3 = go.Figure()

# Add bar chart traces to combined figure
for trace in fig.data:
    fig3.add_trace(trace)

# Add box plot traces with some offset
for i, trace in enumerate(fig2.data):
    # Adjust x positions to place box plots next to bars
    x_positions = []
    for x in trace.x:
        x_positions.append(x)
    trace.x = x_positions
    trace.offsetgroup = "boxplot"
    trace.showlegend = False
    fig3.add_trace(trace)

# Update layout of the combined figure
fig3.update_layout(
    title='Log Price by Room Type Average and Distribution',
    xaxis=dict(title='Room Type'),
    yaxis=dict(title='Log Price'),
    plot_bgcolor='white',
    boxmode='group'
)
```

```
# Show the combined figure
# fig3.show()
```

11 Relationship of Room Type with no of Bedrooms and no of Bathrooms

[19]:
```
# Convert columns to numeric (handling errors)
df['bathrooms'] = pd.to_numeric(df['bathrooms'], errors='coerce')
df['bedrooms'] = pd.to_numeric(df['bedrooms'], errors='coerce')

# Remove rows with missing values in 'room_type', 'bathrooms', and 'bedrooms'
df = df.dropna(subset=['room_type', 'bathrooms', 'bedrooms'])

# Group by 'room_type' and calculate average bathrooms and bedrooms
room_stats = df.groupby('room_type').agg({
    'bathrooms': ['mean', 'count'],
    'bedrooms': ['mean', 'count']
}).reset_index()

# Flatten the multi-index columns
room_stats.columns = ['room_type', 'bathrooms_mean', 'bathrooms_count',␣
 ↪'bedrooms_mean', 'bedrooms_count']
room_stats['bathrooms_mean'] = room_stats['bathrooms_mean'].round(2)
room_stats['bedrooms_mean'] = room_stats['bedrooms_mean'].round(2)

# Create a subplot with 2 rows and 1 column
fig = make_subplots(rows=2, cols=1,
                   subplot_titles=("Average Number of Bathrooms by Room Type",
                                  "Average Number of Bedrooms by Room Type"),
                   vertical_spacing=0.2)

# Add bathrooms bar chart to the first row
bathrooms_bars = go.Bar(
    x=room_stats['room_type'],
    y=room_stats['bathrooms_mean'],
    text=room_stats['bathrooms_mean'],
    textposition='auto',
    name='Bathrooms',
    marker_color=['#1f77b4', '#ff7f0e', '#2ca02c']
)
fig.add_trace(bathrooms_bars, row=1, col=1)

# Add bedrooms bar chart to the second row
bedrooms_bars = go.Bar(
    x=room_stats['room_type'],
    y=room_stats['bedrooms_mean'],
    text=room_stats['bedrooms_mean'],
    textposition='auto',
    name='Bedrooms',
    marker_color=['#1f77b4', '#ff7f0e', '#2ca02c']
)
fig.add_trace(bedrooms_bars, row=2, col=1)

# Add count annotations for bathrooms
for i, row in room_stats.iterrows():
    fig.add_annotation(
        x=row['room_type'],
        y=row['bathrooms_mean'] + 0.1,
        text=f"n={row['bathrooms_count']}",
        showarrow=False,
        row=1, col=1
    )

# Add count annotations for bedrooms
for i, row in room_stats.iterrows():
    fig.add_annotation(
        x=row['room_type'],
        y=row['bedrooms_mean'] + 0.1,
        text=f"n={row['bedrooms_count']}",
        showarrow=False,
        row=2, col=1
    )

# Update layout
fig.update_layout(
    title_text="Relationship Between Room Type V/S Bathrooms and Bedrooms",
    height=700,
    showlegend=False,
    plot_bgcolor='white'
)

# Update axes
fig.update_xaxes(title_text="Room Type", row=1, col=1)
fig.update_xaxes(title_text="Room Type", row=2, col=1)
fig.update_yaxes(title_text="Average Number of Bathrooms", row=1, col=1)
fig.update_yaxes(title_text="Average Number of Bedrooms", row=2, col=1)

# Display the plot
# fig.show()
```

12 Relationship B/W Cancellation Policy and Instant Bookable Status

[20]:
```
# Count the combinations of cancellation_policy and instant_bookable
policy_instant = df.groupby(['cancellation_policy', 'instant_bookable']).size().
↪reset_index(name='count')

# Create a grouped bar chart
fig = px.bar(
    policy_instant,
    x='cancellation_policy',
    y='count',
    color='instant_bookable',
    title='Relationship Between Cancellation Policy and Instant Bookable Status',
    labels={'count': 'Number of Listings', 'cancellation_policy': 'Cancellation␣
↪Policy', 'instant_bookable': 'Instant Bookable'},
    height=500,
    barmode='group'
)

# Calculate percentages for each cancellation policy
total_by_policy = df.groupby('cancellation_policy').size().
↪reset_index(name='total')
policy_instant_pct = pd.merge(policy_instant, total_by_policy,␣
↪on='cancellation_policy')
policy_instant_pct['percentage'] = (policy_instant_pct['count'] /␣
↪policy_instant_pct['total'] * 100).round(1)

# Improve annotation positioning and visibility
for i, row in policy_instant_pct.iterrows():
    # Adjust position based on instant_bookable value
    y_offset = 0.05 * row['count']

    # Create background for better visibility
    fig.add_annotation(
        x=row['cancellation_policy'],
        y=row['count'] + y_offset,
        text=f"{row['percentage']}%",
        showarrow=False,
        font=dict(size=11, color='black'),
        bgcolor='rgba(255, 255, 255, 0.8)',
        bordercolor='rgba(0, 0, 0, 0.3)',
        borderwidth=1,
        borderpad=4
    )

# Update layout
```

```
fig.update_layout(
    xaxis=dict(title='Cancellation Policy'),
    yaxis=dict(title='Number of Listings'),
    plot_bgcolor='white',
    legend_title='Instant Bookable',
    bargap=0.3  # Increase gap between bar groups
)

# Display the plot
# fig.show()
```

13 Dist of Room Types

[21]:
```
# Count occurrences of each room type
room_counts = df['room_type'].value_counts().reset_index()
room_counts.columns = ['room_type', 'count']

# Calculate percentages
total = room_counts['count'].sum()
room_counts['percentage'] = (room_counts['count'] / total * 100).round(1)

# Create a donut chart
fig = go.Figure(data=[go.Pie(
    labels=room_counts['room_type'],
    values=room_counts['count'],
    hole=0.5,
    textinfo='label+percent',
    insidetextorientation='radial',
    texttemplate='%{label}<br>%{percent}',
    marker=dict(colors=px.colors.qualitative.Set2)
)])

# Update layout
fig.update_layout(
    title='Distribution of Room Types',
    annotations=[dict(text='Room Types', x=0.5, y=0.5, font_size=20,␣
 ↪showarrow=False)],
    height=500,
    width=600,
    showlegend=True,
    legend=dict(
        orientation="h",
        yanchor="bottom",
        y=-0.2,
        xanchor="center",
        x=0.5
    )
)

# Add count information in the legend
for i, row in room_counts.iterrows():
    fig.data[0].text = [f"{room_counts.iloc[j]['room_type']}<br>{room_counts.
↪iloc[j]['count']} listings"
                        for j in range(len(room_counts))]

# Show the figure
# fig.show()
```

14 Dist of Bathrooms

[22]:
```
# Convert bathrooms to numeric and handle missing values
df['bathrooms'] = pd.to_numeric(df['bathrooms'], errors='coerce')
df = df.dropna(subset=['bathrooms'])

# Count occurrences of each bathroom count
bathroom_counts = df['bathrooms'].value_counts().reset_index()
bathroom_counts.columns = ['bathrooms', 'count']

# Sort by number of bathrooms
bathroom_counts = bathroom_counts.sort_values('bathrooms')

# Calculate percentages
total = bathroom_counts['count'].sum()
bathroom_counts['percentage'] = (bathroom_counts['count'] / total * 100).round(1)

# Create labels with proper bathroom text
bathroom_counts['label'] = bathroom_counts['bathrooms'].astype(str) + '␣
↪Bathroom' + bathroom_counts['bathrooms'].apply(lambda x: 's' if x > 1 else '')

# Create a column chart with vibrant colors
fig = px.bar(
    bathroom_counts,
    x='label',
    y='count',
    text='count',
    title='Distribution of Bathrooms in Listings',
    labels={'count': 'Number of Listings', 'label': 'Bathroom Count'},
    height=500,
    color='label',  # Color by label to get distinct colors
    color_discrete_sequence=px.colors.qualitative.Bold  # Use a vibrant color␣
↪palette
)
```

```python
# Add percentage annotations with improved visibility
for i, row in bathroom_counts.iterrows():
    fig.add_annotation(
        x=row['label'],
        y=row['count'],
        text=f"{row['percentage']}%",
        showarrow=False,
        yshift=15,
        font=dict(size=11, color='black'),
        bgcolor='rgba(255, 255, 255, 0.9)',
        bordercolor='rgba(0, 0, 0, 0.5)',
        borderwidth=2,
        borderpad=4
    )

# Update layout with more vibrant styling
fig.update_layout(
    title={
        'text': 'Distribution of Bathrooms in Listings',
        'font': {'size': 22, 'color': '#2E4057'},
        'y': 0.95
    },
    xaxis=dict(
        title='Number of Bathrooms',
        tickfont=dict(size=14),
        tickangle=-45
    ),
    yaxis=dict(
        title='Number of Listings',
        tickfont=dict(size=14),
        gridcolor='rgba(220, 220, 220, 0.5)'
    ),
    plot_bgcolor='white',
    showlegend=False,
    height=500,
    bargap=0.3
)

# Show the figure
# fig.show()
```

15 Dist of Bedrooms

[23]:
```python
# Convert bedrooms to numeric and handle missing values
df['bedrooms'] = pd.to_numeric(df['bedrooms'], errors='coerce')
df = df.dropna(subset=['bedrooms'])

# Count occurrences of each bedroom count
bedroom_counts = df['bedrooms'].value_counts().reset_index()
bedroom_counts.columns = ['bedrooms', 'count']

# Sort by number of bedrooms
bedroom_counts = bedroom_counts.sort_values('bedrooms')

# Calculate percentages
total = bedroom_counts['count'].sum()
bedroom_counts['percentage'] = (bedroom_counts['count'] / total * 100).round(1)

# Create labels with proper bedroom text
bedroom_counts['label'] = bedroom_counts['bedrooms'].astype(int).astype(str) + '␣
↪Bedroom' + bedroom_counts['bedrooms'].apply(lambda x: 's' if x > 1 else '')

# Create a column chart
fig = px.bar(
    bedroom_counts,
    x='label',
    y='count',
    text='count',
    title='Distribution of Bedrooms in Listings',
    labels={'count': 'Number of Listings', 'label': 'Bedroom Count'},
    height=500,
    color='bedrooms',
    color_continuous_scale=px.colors.sequential.Viridis
)

# Add percentage annotations
for i, row in bedroom_counts.iterrows():
    fig.add_annotation(
        x=row['label'],
        y=row['count'],
        text=f"{row['percentage']}%",
        showarrow=False,
        yshift=10,
        font=dict(size=10),
        bgcolor='rgba(255, 255, 255, 0.8)',
        bordercolor='rgba(0, 0, 0, 0.3)',
        borderwidth=1,
        borderpad=4
    )

# Update layout
fig.update_layout(
    xaxis=dict(title='Number of Bedrooms'),
    yaxis=dict(title='Number of Listings'),
    plot_bgcolor='white',
    coloraxis_showscale=False
)

# Show the figure
# fig.show()
```

16 Dist of Accommodates

[24]:
```
# Convert accommodates to numeric and handle missing values
df['accommodates'] = pd.to_numeric(df['accommodates'], errors='coerce')
df = df.dropna(subset=['accommodates'])

# Count occurrences of each accommodates value
accommodates_counts = df['accommodates'].value_counts().reset_index()
accommodates_counts.columns = ['accommodates', 'count']

# Sort by number of accommodates
accommodates_counts = accommodates_counts.sort_values('accommodates')

# Calculate percentages
total = accommodates_counts['count'].sum()
accommodates_counts['percentage'] = (accommodates_counts['count'] / total * 100).
 ↪round(1)

# Create labels
accommodates_counts['label'] = accommodates_counts['accommodates'].astype(int).
 ↪astype(str) + ' Person' + accommodates_counts['accommodates'].apply(lambda x:␣
 ↪'s' if x > 1 else '')

# Create a column chart with vibrant colors
fig = px.bar(
    accommodates_counts,
    x='label',
    y='count',
    text='count',
    title='Distribution of Accommodates in Listings',
    labels={'count': 'Number of Listings', 'label': 'Accommodates'},
    height=500,
    color='label',  # Color by label to get distinct colors
    color_discrete_sequence=px.colors.qualitative.Vivid  # Use a vibrant color␣
 ↪palette
)

# Add percentage annotations with improved visibility
for i, row in accommodates_counts.iterrows():
    fig.add_annotation(
        x=row['label'],
        y=row['count'],
        text=f"{row['percentage']}%",
        showarrow=False,
        yshift=15,
        font=dict(size=11, color='black'),
        bgcolor='rgba(255, 255, 255, 0.9)',
        bordercolor='rgba(0, 0, 0, 0.5)',
        borderwidth=2,
        borderpad=4
    )

# Update layout with vibrant styling
fig.update_layout(
    title={
        'text': 'Distribution of Accommodates in Listings',
        'font': {'size': 22, 'color': '#2E4057'},
        'y': 0.95
    },
    xaxis=dict(
        title='Number of People Accommodated',
        tickfont=dict(size=14),
        tickangle=-45
    ),
    yaxis=dict(
        title='Number of Listings',
        tickfont=dict(size=14),
        gridcolor='rgba(220, 220, 220, 0.5)'
    ),
    plot_bgcolor='white',
    showlegend=False,
    height=500,
    bargap=0.3
)

# Show the figure
# fig.show()
```

```
[25]: df.to_csv(r'D:\Quadratic\cleaned_airbnb_data2.csv', index=False)
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: