Predict the house value on Airbnb.

CAC → Customer Acq Cost
LTV → Lifetime value customer

$\frac{CAC}{LTV} \Rightarrow \downarrow$ $\frac{LTV}{CAC} \uparrow$

Traditionally : https://medium.com/swlh/diligence-at-social-capital-part-3-cohorts-and-revenue-ltv-ab65a07464e1

Blog : https://medium.com/airbnb-engineering/using-machine-learning-to-predict-value-of-homes-on-airbnb-9272d3d4739d

Functional Requirements

1. Accurately predict the home value
2. Real-time predict & Batch prediction { + daily prediction update }
   ⇓
   API
3. Explainability
   ↳ SHAP values { "proximity to the beach could increase property rate by 12%"
4. Provide actionable recommendations
   ↳ Adding wifi to increase rent by ₹1000/night

5. A/B Testing frame

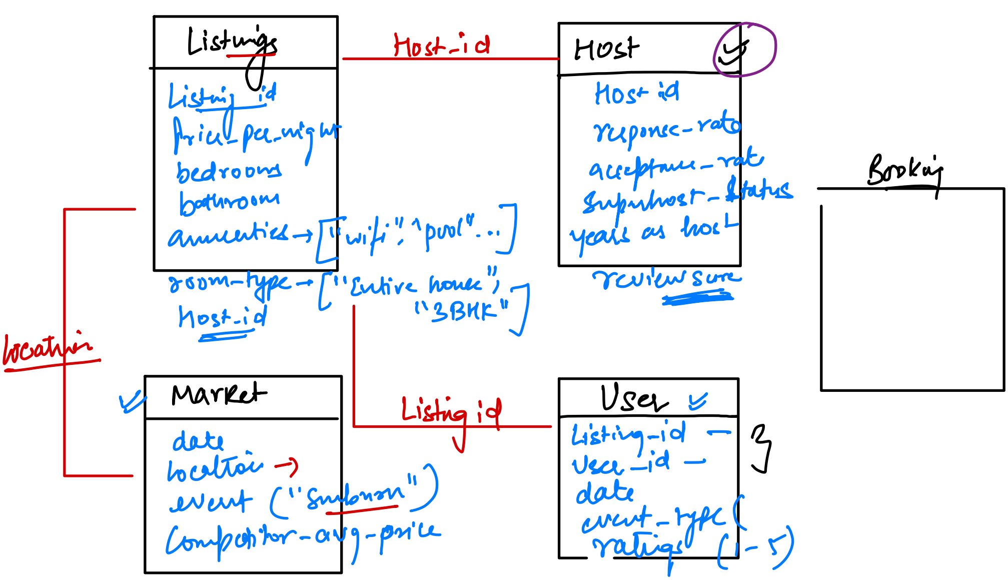
Impact of model on business metrics. (booking conversions, revenue)

:

Non-functional Requirements

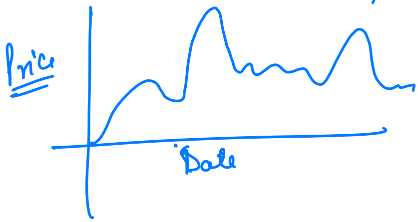
1. Latency → <100ms for real-time prediction
2. Scalable → 10Mn listing, 10K
3. Reliability → 99.99% uptime → realtime preds.
4. Maintainability → Model retraining should be minimal.
5. Privacy → GDPR - compliant data handling

Data Overview



Key Features

1. Historical demand → Bookings data for last 1 year.



2. Host Quality → [Avg responce time], review score

3. Location Score → Distance to Landmarks.

Red fort

Listing id

lat / long

Haversine formula.

```
from geopy.distance import great_circle
def compute_distance(row, landmark_coords):
    return great_circle((row['latitude'], row['longitude']), landmark_coords).miles
df['distance_to_downtown'] = df.apply(compute_distance, args=(downtown_coords,), axis=1)
```

listing	...	Distance	Distance wrt center
—		{ "Red fort": 30 Qutub minar: 40 } center: —	—

Preprocessing

① Handle Missing value

We need to check if any data is missing, and whether that data is missing at random. If not, we need to investigate why and understand the root cause. If yes, we should impute the missing values.

Median → Numerical

Simple Imputer
KNN Imputer

2. Encoding

- **Encoding Categorical Variables**: Often we cannot use the raw categories in the model, since the model doesn't know how to fit on strings. When the number of categories is low, we may consider using one-hot encoding. However, when the cardinality is high, we might consider using ordinal encoding, encoding by frequency count of each category.

/ embedding

+ Visualization
+ Bi-Multi-variate analysis (Pandas, Seaborn)

Hypothesis Testing

Yes/No 1/0

Hypothesis 1 : Listings with 24-hour checkin feature results in 15% higher booking

	Yes	No
1 →	20	21
2 →	30	30
3 →	40	40
4 →	15	50
	⋮	⋮

In a year, if i use the overall booking for properties that say yes

→ T-test → Significant difference ✓

Confidence intervals → Yes [LR, H·R]

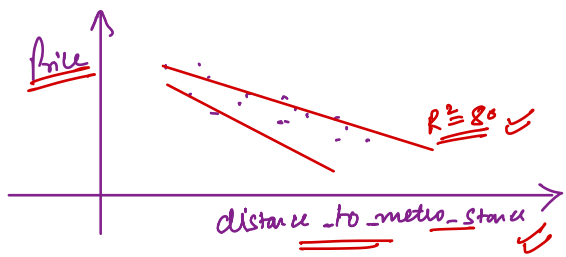
NO [LR, HR]

Hypothesis 2:

Proximity to Public transport facility would explain the 25% price variable in Urban areas

Test

Linear Regression model



$R^2 = 0.18$

Hypothesis 3: "Hosts with a 90%+ response rate achieve 20% higher revenue."

Test: ANOVA across response rate buckets (0-50%, 50-90%, 90-100%).

Result: Significant F-statistic ($p < 0.05$), revenue peaks at 90-100%.

Modelling

	Pros	Cons	RMSE
1. Linear Regression	interpretable / easy / fast	Misses non linear trend	110 USD
2. XGBOOST	Robust to outliers / interaction	Requires tuning	85 USD ✓
3. DNN	Complex pattern	High latency, hard to debug.	90 USD

ML System Architecture

Dataset → Listings / Host / Bookings

1. Ingestion

→ SQL → Spark / Dask

→ Kafka streams → Spark Structured Stream

→ S3 (CSV / parquet) → spark

2. Storage

Real-time feature → Redis

Historical → S3

③ Training pipeline

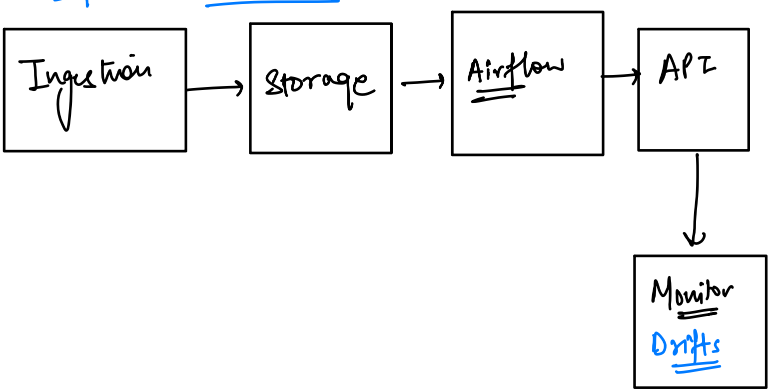
↳ Airflow DAG for triggering weekly runs.

④ Real-time API

↳ Flask | FastAPI

```python
from fastapi import FastAPI
app = FastAPI()
@app.post("/predict")
async def predict(listing_id: str):
    features = redis_client.get(listing_id)  # Fetch precomputed features
    prediction = xgb_model.predict([features])[0]
    return {"predicted_value": prediction}
```

System Architecture



SHAP Values

```python
import shap
explainer = shap.TreeExplainer(xgb_model)
shap_values = explainer.shap_values(X_test)
```

shap.summary_plot(   )

Amenities

Distance to metro

```python
# 1. Load Data
df = spark.read.parquet("s3://airbnb/listings")

# 2. Compute Target Variable (e.g., revenue)
df = df.withColumn("revenue_last_6mo", F.sum("price").over(Window.partitionBy("listing_id").ran
geBetween(-180, 0)))

# 3. Preprocess
imputer = Imputer(inputCols=["response_rate"], outputCols=["response_rate"], strategy="median")
df = imputer.fit(df).transform(df)

# 4. Train XGBoost
from xgboost import XGBRegressor
xgb = XGBRegressor(max_depth=8, learning_rate=0.1)
xgb.fit(X_train, y_train)

# 5. SHAP Analysis
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.plots.waterfall(shap_values[0])  # Explain one prediction
```

✓ { Informatica } { Alteryx } { Dataiku DSS }