# Airbnb Dynamic Pricing Recommendation Engine — Full Package

Generated: Detailed deliverables including full code, API server, Tableau fields, and PDF generator.

## 1. Overview

This document contains the complete code, deployment instructions, Tableau calculated fields, and PDF report generator for the Airbnb Dynamic Pricing Recommendation Engine. Use the code blocks as copy-paste-ready files. Follow 'How to run' sections for local execution.

## 2. pricing_engine.py (Full script)

```
# pricing_engine.py
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, mean_absolute_error
from xgboost import XGBRegressor
import joblib
import os
from datetime import datetime

RANDOM_STATE = 42

def load_data(path):
    df = pd.read_csv(path, parse_dates=['date'], dayfirst=True, low_memory=False)
    return df

def clean_basic(df):
    def to_num(x):
        if pd.isna(x): return np.nan
        s = str(x).replace('$','').replace(',','').strip()
        return float(s) if s!='' else np.nan
    for c in ['price','cleaning_fee','security_deposit','competitor_price_avg']:
        if c in df.columns:
            df[c] = df[c].astype(str).replace('nan','', regex=False).apply(lambda v: np.nan if v=='' els
            df[c] = df[c].apply(lambda v: to_num(v) if pd.notna(v) else np.nan)
    if 'amenities' in df.columns:
        df['amenities_count'] = df['amenities'].fillna('').apply(lambda s: s.count(',')+1 if s.strip() e
    if 'host_since' in df.columns:
        df['host_since'] = pd.to_datetime(df['host_since'], errors='coerce')
        df['host_tenure_days'] = (pd.Timestamp.now() - df['host_since']).dt.days.fillna(0)
    if 'availability_365' in df.columns:
        df['occupancy_estimate'] = 1.0 - (df['availability_365'].fillna(365) / 365.0)
    if 'date' in df.columns:
        df['month'] = df['date'].dt.month
        df['dayofweek'] = df['date'].dt.dayofweek
        df['is_weekend'] = df['dayofweek'].isin([5,6]).astype(int)
    numeric_cols = df.select_dtypes(include=['float64','int64']).columns.tolist()
    for c in numeric_cols:
        df[c] = pd.to_numeric(df[c], errors='coerce')
    return df

def build_pipeline(categorical_cols, numeric_cols):
    cat_pipe = Pipeline([
        ('impute', SimpleImputer(strategy='constant', fill_value='unknown')),
        ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False))
    ])
    num_pipe = Pipeline([
        ('impute', SimpleImputer(strategy='median')),
        ('scale', StandardScaler())
    ])
    pre = ColumnTransformer([
        ('cat', cat_pipe, categorical_cols),
        ('num', num_pipe, numeric_cols),
    ], remainder='drop')
    # Store attributes for later use
    pre.categorical_features = categorical_cols
    pre.numeric_features = numeric_cols
    return pre
def train_model(df, model_path='model.joblib', preprocess_path='preprocessor.joblib'):
    df = df.copy()
    df = df[df['price'].notna()]
    df['log_price'] = np.log1p(df['price'])
    categorical_cols = ['city','neighbourhood','property_type','room_type']
    categorical_cols = [c for c in categorical_cols if c in df.columns]
    numeric_cols = ['accommodates','bedrooms','beds','bathrooms',
                    'amenities_count','review_scores_rating','number_of_reviews',
                    'reviews_per_month','host_tenure_days','occupancy_estimate',
                    'month','dayofweek','is_weekend','competitor_price_avg']
    numeric_cols = [c for c in numeric_cols if c in df.columns]
    X = df[categorical_cols + numeric_cols].copy()
    y = df['log_price'].values
    pre = build_pipeline(categorical_cols, numeric_cols)
    if 'date' in df.columns:
        df_sorted = df.sort_values('date')
        split_index = int(len(df_sorted) * 0.8)
        train_idx = df_sorted.index[:split_index]
        test_idx = df_sorted.index[split_index:]
        X_train, X_test = X.loc[train_idx], X.loc[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]
    else:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=RANDOM_STA
    X_train_t = pre.fit_transform(X_train)
    X_test_t = pre.transform(X_test)
    xgb = XGBRegressor(objective='reg:squarederror', random_state=RANDOM_STATE, n_jobs=4)
    param_grid = {'n_estimators':[100,200], 'max_depth':[4,6], 'learning_rate':[0.05,0.1]}
    grid = GridSearchCV(xgb, param_grid, cv=3, scoring='neg_root_mean_squared_error', verbose=1, n_jobs=
    grid.fit(X_train_t, y_train)
    best = grid.best_estimator_
    preds_log = best.predict(X_test_t)
    preds_price = np.expm1(preds_log)
    y_test_price = np.expm1(y_test)
    rmse = mean_squared_error(y_test_price, preds_price, squared=False)
    mae = mean_absolute_error(y_test_price, preds_price)
    print(f"Model trained. Test RMSE (raw price): {rmse:.2f}, MAE: {mae:.2f}")
    joblib.dump(best, model_path)
    joblib.dump(pre, preprocess_path)
    return {
        'model': best,
        'preprocessor': pre,
        'categorical_cols': categorical_cols,
        'numeric_cols': numeric_cols,
        'rmse': rmse,
        'mae': mae
    }

def predict_price_for_df(df_in, artifacts):
    pre = artifacts['preprocessor']
    model = artifacts['model']
    X = df_in[artifacts['categorical_cols'] + artifacts['numeric_cols']].copy()
    X_t = pre.transform(X)
    preds_log = model.predict(X_t)
    preds_price = np.expm1(preds_log)
    return preds_price

def suggest_price(row, predicted_price, slider_percent=0.0, min_price_col=None, max_multiplier=1.6):
    base = predicted_price * (1.0 + slider_percent)
    floors = []
    if min_price_col and min_price_col in row and pd.notna(row[min_price_col]):
        floors.append(float(row[min_price_col]))
    if 'price' in row and pd.notna(row['price']):
        floors.append(float(row['price'])*0.5)
    floors.append(predicted_price*0.6)
    min_price = max(floors) if floors else predicted_price*0.6
    max_price = predicted_price * max_multiplier
    if 'review_scores_rating' in row and pd.notna(row['review_scores_rating']):
        if row['review_scores_rating'] < 75 and base > predicted_price * 1.1:
            base = predicted_price * 1.1
    suggested = float(np.clip(base, min_price, max_price))
    return round(suggested, 2)

def run_full_training(csv_path, artifacts_dir='artifacts'):
    os.makedirs(artifacts_dir, exist_ok=True)
    df = load_data(csv_path)
    df = clean_basic(df)
    artifacts = train_model(df, model_path=os.path.join(artifacts_dir,'model.joblib'),
                            preprocess_path=os.path.join(artifacts_dir,'preproc.joblib'))
    predicted_prices = predict_price_for_df(df, artifacts)
    df['predicted_price'] = np.round(predicted_prices, 2)
    df['suggested_price'] = df.apply(lambda r: suggest_price(r, r['predicted_price'], slider_percent=0.0
    out_csv = os.path.join(artifacts_dir, 'price_suggestions.csv')
    df.to_csv(out_csv, index=False)
    print(f"Saved suggestions to {out_csv}")
    return artifacts, df

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser(description="Train Airbnb Pricing Engine")
    parser.add_argument('--data', required=True, help='path to historical data CSV')
    parser.add_argument('--out', default='artifacts', help='output artifacts folder')
    args = parser.parse_args()
    artifacts, df = run_full_training(args.data, artifacts_dir=args.out)
```

## 3. FastAPI server (app.py)

```
# app.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from typing import List, Optional
import joblib
import numpy as np
import pandas as pd
import uvicorn
import os

ARTIFACTS_DIR = os.environ.get('ARTIFACTS_DIR','artifacts')
MODEL_PATH = os.path.join(ARTIFACTS_DIR,'model.joblib')
PREPROC_PATH = os.path.join(ARTIFACTS_DIR,'preproc.joblib')

model = joblib.load(MODEL_PATH)
preproc = joblib.load(PREPROC_PATH)

app = FastAPI(title="Airbnb Pricing Engine API")

class PredictRequest(BaseModel):
    listing_id: Optional[str]
    city: Optional[str] = None
    neighbourhood: Optional[str] = None
    property_type: Optional[str] = None
    room_type: Optional[str] = None
    accommodates: Optional[float] = None
    bedrooms: Optional[float] = None
    beds: Optional[float] = None
    bathrooms: Optional[float] = None
    amenities_count: Optional[float] = None
    review_scores_rating: Optional[float] = None
    number_of_reviews: Optional[float] = None
    reviews_per_month: Optional[float] = None
    host_tenure_days: Optional[float] = None
    occupancy_estimate: Optional[float] = None
    month: Optional[int] = None
    dayofweek: Optional[int] = None
    is_weekend: Optional[int] = None
    competitor_price_avg: Optional[float] = None
    price: Optional[float] = None
    min_nightly_price: Optional[float] = None
    slider_percent: Optional[float] = Field(0.0, description="e.g. 0.1 for +10%, -0.2 for -20%")

class BatchRequest(BaseModel):
    rows: List[PredictRequest]

def _to_dataframe(req: PredictRequest):
    d = req.dict()
    return pd.DataFrame([d])

def _predict_from_df(df: pd.DataFrame):
    try:
        X_t = preproc.transform(df)
    except Exception as e:
        raise HTTPException(status_code=400, detail=f"Preprocessor transform failed: {e}")
    preds_log = model.predict(X_t)
    preds = np.expm1(preds_log)
    return preds

def suggest_price_from_row(row: pd.Series, predicted_price: float, slider_percent: float=0.0, max_multip
    base = predicted_price * (1.0 + slider_percent)
    floors = []
    if 'min_nightly_price' in row and pd.notna(row.get('min_nightly_price')):
        floors.append(float(row['min_nightly_price']))
    if 'price' in row and pd.notna(row.get('price')):
        floors.append(float(row['price']) * 0.5)
    floors.append(predicted_price * 0.6)
    min_price = max(floors) if floors else predicted_price * 0.6
    max_price = predicted_price * max_multiplier
    if 'review_scores_rating' in row and pd.notna(row.get('review_scores_rating')):
        if row['review_scores_rating'] < 75 and base > predicted_price * 1.1:
            base = predicted_price * 1.1
    suggested = float(np.clip(base, min_price, max_price))
    return round(suggested, 2)

@app.post("/predict")
def predict(req: PredictRequest):
    df = _to_dataframe(req)
    preds = _predict_from_df(df)
    predicted_price = float(preds[0])
    suggested = suggest_price_from_row(df.iloc[0], predicted_price, slider_percent=req.slider_percent or
    return {
        "listing_id": req.listing_id,
        "predicted_price": round(predicted_price, 2),
        "suggested_price": suggested,
        "slider_percent": req.slider_percent
    }

@app.post("/batch_predict")
def batch_predict(req: BatchRequest):
    rows = [r.dict() for r in req.rows]
    df = pd.DataFrame(rows)
    preds = _predict_from_df(df)
    results = []
    for i, r in df.iterrows():
        p = float(preds[i])
        slider = float(r.get('slider_percent', 0.0) or 0.0)
        suggested = suggest_price_from_row(r, p, slider_percent=slider)
        results.append({
            "index": int(i),
            "predicted_price": round(p,2),
            "suggested_price": suggested,
            "slider_percent": slider
        })
    return {"results": results}

if __name__ == "__main__":
    uvicorn.run("app:app", host="0.0.0.0", port=8000, reload=True)
```

## 4. Tableau: Parameters & Calculated Fields

Parameter: price_adjustment - Data type: Float - Current value: 0.0 - Range: Min -0.20, Max 0.30, Step 0.01 Calculated Field: Suggested Price (raw) [Predicted_Price] * (1 + [price_adjustment]) Calculated Field: Suggested Price (clipped) FLOAT( IF [Review_Scores_Rating] < 75 AND [Suggested Price (raw)] > [Predicted_Price] * 1.1 THEN MIN( [Predicted_Price] * 1.1, [Predicted_Price] * 1.6 ) ELSE MIN( [Predicted_Price] * 1.6, MAX( IFNULL([min_nightly_price], 0 ), [Predicted_Price] * 0.6, IFNULL([price],0) * 0.5, [Suggested Price (raw)] ) ) END ) Calculated Field: Opportunity % IFNULL([price],0) = 0 THEN 0 ELSE ([Suggested Price (clipped)] - [price]) / [price] END

## 5. PDF Report Generator (generate_report.py)

```
# generate_report.py
import pandas as pd
import matplotlib.pyplot as plt
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from reportlab.lib.utils import ImageReader
import os

def plot_top_cities(df, outpath='top_cities.png'):
    agg = df.groupby('city').agg({'price':'mean','predicted_price':'mean','suggested_price':'mean'}).sor
    ax = agg[['price','predicted_price','suggested_price']].plot(kind='bar', figsize=(10,4))
    ax.set_title("Top 10 Cities - Avg Prices")
    plt.tight_layout()
    plt.savefig(outpath)
    plt.close()
    return outpath

def plot_residuals(df, outpath='residuals.png'):
    df = df[df['price'].notna()]
    df['residual'] = df['predicted_price'] - df['price']
    ax = df['residual'].hist(bins=50, figsize=(8,4))
    ax.set_title('Residuals (Predicted - Actual)')
    plt.tight_layout()
    plt.savefig(outpath)
    plt.close()
    return outpath

def create_pdf_report(csv_path='artifacts/price_suggestions.csv', pdf_path='price_report.pdf'):
    df = pd.read_csv(csv_path)
    city_plot = plot_top_cities(df)
    resid_plot = plot_residuals(df)

    c = canvas.Canvas(pdf_path, pagesize=A4)
    width, height = A4

    c.setFont("Helvetica-Bold", 20)
    c.drawCentredString(width/2, height-100, "Airbnb Pricing Suggestions Report")
    c.setFont("Helvetica", 12)
    c.drawString(50, height-130, f"Generated: {pd.Timestamp.now().strftime('%Y-%m-%d %H:%M')}")
    c.drawString(50, height-150, f"Total listings analyzed: {len(df)}")
    c.showPage()

    c.setFont("Helvetica-Bold", 16)
    c.drawString(50, height-60, "Top Cities: Average Prices")
    img = ImageReader(city_plot)
    c.drawImage(img, 50, height-500, width=500, height=350, preserveAspectRatio=True)
    c.showPage()

    c.setFont("Helvetica-Bold", 16)
    c.drawString(50, height-60, "Model Residuals")
    img = ImageReader(resid_plot)
    c.drawImage(img, 50, height-500, width=500, height=350, preserveAspectRatio=True)
    c.showPage()

    c.setFont("Helvetica-Bold", 16)
    c.drawString(50, height-60, "Top 10 Upsell Opportunities (Suggested > Current)")
    opportunities = df[df['price'].notna()].copy()
    opportunities['opp_pct'] = (opportunities['suggested_price'] - opportunities['price']) / opportuniti
    opportunities = opportunities.sort_values('opp_pct', ascending=False).head(10)
    y = height - 100
    c.setFont("Helvetica", 10)
    for _, row in opportunities.iterrows():
        line = f"{str(row.get('listing_id',''))[:15]:15} | {str(row.get('city',''))[:15]:15} | Current:
        c.drawString(50, y, line)
        y -= 14
        if y < 80:
            c.showPage()
            y = height - 80
    c.save()
    print(f"Saved PDF: {pdf_path}")
    return pdf_path

if __name__ == '__main__':
    create_pdf_report()
```

## 6. How to run everything locally

1) Install dependencies: pip install pandas numpy scikit-learn xgboost joblib matplotlib reportlab fastapi uvicorn 2) Train model and create suggestions CSV: python pricing_engine.py --data path/to/historical_airbnb.csv --out artifacts 3) Start FastAPI: Ensure artifacts/model.joblib and artifacts/preproc.joblib are present. uvicorn app:app --host 0.0.0.0 --port 8000 4) Generate PDF report: python generate_report.py