InternnCraft Data Science Task One Report

Muhammad Ahmad Naveed

Report: Real Estate Data Analysis and Model Evaluation

Introduction

The objective of this task was to clean and preprocess the real estate data and then evaluate it using different models to predict property prices. The process followed the steps of data cleaning and preprocessing, feature engineering, and model training using the models of Linear Regression, Random Forest, and Gradient Boosting Regressors.

Data Cleaning and Preprocessing

1. Missing Values:

    Missing values from the columns agency and agent were replaced with NA.

    For numerical features, their averages were taken where applicable.

2. Data Standardization:

    Textual columns were standardized by removing whitespaces and by converting the text to lowercase.

    The dataset was then filtered to keep only the latitude, longitude, price, baths, and bedrooms values.

3. Outlier Detection and Treatment:

    Outliers were also identified using the interquartile range of different values. The most significant outliers were then winsorized to reduce their influence on the analysis.

    After that, extreme values were replaced with their respective means where necessary to diminish their skewedness.

4. Feature Engineering:

    Added features like house_age, priceToSqft, bedroomToFloor, and bathToBedroom to improve performance.

    The area values were standardized to square feet for consistency, and category variables were encoded using one-hot encoding.

Exploratory Data Analysis

A folium map was also generated for the visualization of the distribution of properties by price, location, number of bedrooms, and baths.

The relationships between features such as area size, bedrooms, and price were then analysed using derived metrics like priceToSqft.

Model Evaluation

I trained and evaluated three models:

1. Linear Regression:

   MAE: 7124822.044877812

   MSE: 92482556338327.88

   R2: 0.24622447667397995

   This model showed moderate performance with some underfitting.

2. Random Forest Regressor:

   MAE: 4124486.804822419

   MSE: 49980837246734.59

   R2: 0.5926331057058325

   Performance improved significantly with this since it also handled non-linear relationships better.

3. Gradient Boosting Regressor:

   MAE: 5580709.0079267025

   MSE: 62483769795104.34

   R2: 0.4907284341883179

   This model offered a good balance between the bias and the variance, but the Random Forest Regressor performed better overall.

Findings

Outlier Handling: Treating the outliers significantly improved the models' accuracy and robustness.

Feature Importance: Area size, location, and the number of bedrooms and baths were the most impactful features in predicting the prices.

Model Performance: The Random Forest Regressor provided the highest accuracy among the three models evaluated. This made it more suitable for this dataset, particularly when capturing the non-linear relationships.

Recommendations

Model: Random Forest Regressor, based on its R^2 score and error metrics.

Further Feature Engineering: Additional location-specific features like area facilities and safety, etcetera would further enhance the precision of the price predictions.

Data Collection and Consistency: Ensuring consistent and uniform data entries would greatly reduce the preprocessing time and also improve the model's performance.

Code

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import folium

import datetime

from scipy import stats

from scipy.stats import zscore

from scipy.stats.mstats import winsorize

from sklearn.preprocessing import LabelEncoder

from folium.plugins import MarkerCluster

from IPython.display import display, HTML

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

pd.options.display.max_columns = None

df = pd.read_csv('zameen-updated.csv')
```

```
missing_values = df.isnull().sum()
print(missing_values)
df['agency'].fillna('NA', inplace=True)
df['agent'].fillna('NA', inplace=True)
df.isnull().sum()
categories = ['page_url', 'property_type', 'location', 'city', 'province_name', 'purpose']
for c in categories:
 df[c] = df[c].str.strip().str.lower()
df = df[df['price'] > 0]
df = df[(df['latitude'].between(-90, 90)) & (df['longitude'].between(-180, 180))]
df = df[df['baths'] >= 0]
df = df[df['bedrooms'] >= 0]
duplicates = df.duplicated().sum()
print(duplicates)
nc = ['price', 'latitude', 'longitude', 'baths', 'bedrooms', 'Area Size']
count = {}
for c in nc:
    q1 = df[c].quantile(0.25)
    q3 = df[c].quantile(0.75)
    iqr = q3 - q1
    lb = q1 - 1.5 * iqr
    ub = q3 + 1.5 * iqr
    outlier = (df[c] < lb) | (df[c] > ub)
    count[c] = outlier.sum()
print(count)
df['price'] = winsorize(df['price'], limits=[0.01, 0.01])
df['latitude'] = winsorize(df['latitude'], limits=[0.05, 0.05])
df['longitude'] = winsorize(df['longitude'], limits=[0.05, 0.05])
df['baths'] = winsorize(df['baths'], limits=[0.05, 0.05])
```

```
df['bedrooms'] = winsorize(df['bedrooms'], limits=[0.05, 0.05])
df['Area Size'] = winsorize(df['Area Size'], limits=[0.05, 0.05])
price_q1 = df['price'].quantile(0.25)
price_q3 = df['price'].quantile(0.75)
price_iqr = price_q3 - price_q1
price_lb = price_q1 - 1.5 * price_iqr
price_ub = price_q3 + 1.5 * price_iqr
price_mean = df['price'].mean()
df.loc[df['price'] > price_ub, 'price'] = price_mean
df.loc[df['price'] < price_lb, 'price'] = price_mean
outlierUpdated = (df[nc] < price_lb) | (df[nc] > price_ub)
updatedCountOutlier = outlierUpdated.sum()
print(updatedCountOutlier)
df['agency'] = df['agency'].str.strip().str.title()
df['agency'].fillna('NA', inplace=True)
lat_mean = df['latitude'].mean()
lon_mean = df['longitude'].mean()
m = folium.Map(location=[lat_mean, lon_mean], zoom_start=6)
marker_cluster = MarkerCluster().add_to(m)
for _, row in df.iterrows():
 folium.Marker(
 location=[row['latitude'], row['longitude']],
 popup=f"Price: {row['price']}, Beds: {row['bedrooms']}, Baths: {row['baths']}"
 ).add_to(marker_cluster)
map_filename = 'house_price_map.html'
m.save(map_filename)
from datetime import datetime
currYear = datetime.now().year
df['date_added'] = pd.to_datetime(df['date_added'], errors='coerce')
```

```
df['house_age'] = currYear - df['date_added'].dt.year // 365
df['house_age'].fillna(df['house_age'].median(), inplace=True)
def convert_area(area):
 if isinstance(area, str):
     area = area.replace(',', '')
 if 'Marla' in area:
     value = float(area.split()[0])
     return value * 272.25
 elif 'Kanal' in area:
     value = float(area.split()[0])
     return value * 20 * 272.25
 elif 'Square Feet' in area:
     value = float(area.split()[0])
     return value
 return area
df['area'] = df['area'].apply(convert_area)
df['area'] = df['area'].astype(str)
df['area'] = df['area'].str.replace(' Marla', '', regex=False).str.replace(',', '').astype(float)
print(df[['area']].head())
df['priceToSqft'] = df['price'] / df['area']
df['bedroomToFloor'] = df['bedrooms'] / df['area']
df['bedroomToFloor'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['bedroomToFloor'].fillna(df['bedroomToFloor'].median(), inplace=True)
df['bathToBedroom'] = df['baths'] / df['bedrooms']
df['bathToBedroom'].replace([np.inf, -np.inf], np.nan, inplace=True)
df['bathToBedroom'].fillna(df['bathToBedroom'].median(), inplace=True)
print(df.columns)
categorical_columns = ['property_type', 'location', 'city', 'province_name', 'purpose', 'agency',
'agent', 'Area Type', 'Area Category']
```

```
df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=True)
df['price_zscore'] = zscore(df['price'])
outlierHigh = df[df['price_zscore'] > 3]
outlierLow = df[df['price_zscore'] < -3]
features = ['area', 'bedrooms', 'baths', 'house_age', 'latitude', 'longitude']
target = 'price'
X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
lrModel = LinearRegression()
lrModel.fit(X_train, y_train)
yPredLR = lrModel.predict(X_test)
maeLR = mean_absolute_error(y_test, yPredLR)
mseLR = mean_squared_error(y_test, yPredLR)
r2LR = r2_score(y_test, yPredLR)
print("MAE:", maeLR)
print("MSE:", mseLR)
print("R2:", r2LR)
rfModel = RandomForestRegressor(n_estimators=100, random_state=42)
rfModel.fit(X_train, y_train)
yPredRF = rfModel.predict(X_test)
maeRF = mean_absolute_error(y_test, yPredRF)
mseRF = mean_squared_error(y_test, yPredRF)
r2RF = r2_score(y_test, yPredRF)
print("MAE:", maeRF)
print("MSE:", mseRF)
print("R2:", r2RF)
gbModel = GradientBoostingRegressor(n_estimators=100, random_state=42)
gbModel.fit(X_train, y_train)
```

```
yPredGB = gbModel.predict(X_test)
maeGB = mean_absolute_error(y_test, yPredGB)
mseGB = mean_squared_error(y_test, yPredGB)
r2GB = r2_score(y_test, yPredGB)
print("MAE:", maeGB)
print("MSE:", mseGB)
print("R2:", r2GB)
```