22p-9295-amber-assign2

May 6, 2024

[279]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Load dataset
df = pd.read_csv('listing_data_publish.csv')
```

[280]:

```
df = df.dropna()
```

[281]:

```
df = df.drop_duplicates()
```

[282]:

```
df = df.fillna(df.mean())
```

[283]:

```
q1 = df.quantile(0.25)
q3 = df.quantile(0.75)
IQR = q3 - q1
print("Inter Quantile Range = ", IQR)
```

```
Inter Quantile Range =  listing_id           NaN
type                 NaN
sub_type             NaN
start_date           NaN
end_date             NaN
listing_type         NaN
building_age         NaN
total_floor_count    NaN
floor_no             NaN
room_count           NaN
size                 NaN
student_avaliable    NaN
mortgage_avaliable   NaN
address              NaN
furnished            NaN
```

```
heating_type          NaN
price                 NaN
currency              NaN
dtype: float64
```

[284]:
```
upperBound = q1 - 1.5 * IQR
print("Upper Bound: ", upperBound)
```

```
Upper Bound:  listing_id           NaN
type                  NaN
sub_type              NaN
start_date            NaN
end_date              NaN
listing_type          NaN
building_age          NaN
total_floor_count     NaN
floor_no              NaN
room_count            NaN
size                  NaN
student_avaliable     NaN
mortgage_avaliable    NaN
address               NaN
furnished             NaN
heating_type          NaN
price                 NaN
currency              NaN
dtype: float64
```

[285]:
```
lowerBound = q3 + 1.5 * IQR
print("Lower Bound: ", lowerBound)
```

```
Lower Bound:  listing_id           NaN
type                  NaN
sub_type              NaN
start_date            NaN
end_date              NaN
listing_type          NaN
building_age          NaN
total_floor_count     NaN
floor_no              NaN
room_count            NaN
size                  NaN
student_avaliable     NaN
mortgage_avaliable    NaN
address               NaN
furnished             NaN
heating_type          NaN
price                 NaN
```

```
currency             NaN
dtype: float64
```

[286]:

```
removeOutliers = df[(df >= lowerBound) & (df <= upperBound)]
print("Removing Outliers = ", removeOutliers)
removingNullVal = df.fillna(df.mean())
print("Removing NULL Values after removing the outliers = ", removingNullVal)
```

```
Removing Outliers =  Empty DataFrame
Columns: [listing_id, type, sub_type, start_date, end_date, listing_type,
building_age, total_floor_count, floor_no, room_count, size, student_avaliable,
mortgage_avaliable, address, furnished, heating_type, price, currency]
Index: []
Removing NULL Values after removing the outliers =  Empty DataFrame
Columns: [listing_id, type, sub_type, start_date, end_date, listing_type,
building_age, total_floor_count, floor_no, room_count, size, student_avaliable,
mortgage_avaliable, address, furnished, heating_type, price, currency]
Index: []
```

[287]:

```
# Feature Engineering
le = LabelEncoder()
df['address'] = le.fit_transform(df['address'])
le = LabelEncoder()
df['type'] = le.fit_transform(df['type'])
le = LabelEncoder()
df['sub_type'] = le.fit_transform(df['sub_type'])
```

[288]:

```
# Convert 'start_date' to datetime and extract numerical columns
df['start_date'] = pd.to_datetime(df['start_date'])
df['start_day'] = df['start_date'].dt.day
df['start_month'] = df['start_date'].dt.month
df['start_year'] = df['start_date'].dt.year
df = df.drop(['start_date'], axis=1)
```

[289]:

```
# Convert 'start_date' to datetime and extract numerical columns
df['end_date'] = pd.to_datetime(df['end_date'])
df['end_day'] = df['end_date'].dt.day
df['end_month'] = df['end_date'].dt.month
df['end_year'] = df['end_date'].dt.year
df = df.drop(['end_date'], axis=1)
```

[290]:

```
df.columns
```

[290]:

```
Index(['listing_id', 'type', 'sub_type', 'listing_type', 'building_age',
       'total_floor_count', 'floor_no', 'room_count', 'size',
       'student_avaliable', 'mortgage_avaliable', 'address', 'furnished',
       'heating_type', 'price', 'currency', 'start_day', 'start_month',
       'start_year', 'end_day', 'end_month', 'end_year'],
      dtype='object')
```

[291]:

```
# Split data into training and testing sets
# z=df.drop(['start_day', 'start_month','start_year' ], axis=1)
X = df.drop(['price', 'currency'], axis=1)
y = df['price']
# Check if the dataset is not empty
if len(X) > 0 and len(y) > 0:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,␣
  ↪random_state=42)
else:
    print("Error: The dataset is empty.")
```

```
Error: The dataset is empty.
```

[294]:

```
# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[ ]:

```
# Build ANN model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

```
C:\Users\HP\AppData\Roaming\Python\Python312\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[293]:

```
# Train the model
model.fit(X_train_scaled, y_train, epochs=100, batch_size=32,␣
  ↪validation_data=(X_test_scaled, y_test))
```

[ ]:

```
# Evaluate the model
mse = mean_squared_error(y_test, model.predict(X_test_scaled))
mae = mean_absolute_error(y_test, model.predict(X_test_scaled))
print(f'Test MSE: {mse}, Test MAE: {mae}')
```