Artificial intelligence _house predicting of process using machine learning

Phase3 -Development Part-1

ABSTRACT :

The primary objectives of this project are to enhance the accuracy of house price predictions, assist homebuyers and sellers in making informed decisions, and contribute to the real estate industry's efficiency. The research entails data collection, preprocessing, feature engineering, model selection, training, and evaluation. We also emphasize model interpretability, allowing users to understand the driving factors behind each prediction.

MODULES:

●**NumPy**: NumPy is fundamental for handling numerical operations and working with arrays. It's crucial for data manipulation and preprocessing.

XGBoost or LightGBM: These are gradient boosting libraries that can significantly improve the performance of regression models.

Flask or Django: If you plan to create a web application for users to interact with your model, these Python web frameworks are popular choices.

Jupyter Notebook: Jupyter Notebook is an interactive environment for data analysis and model development. It's great for documenting and sharing your work.

Joblib or Pickle: These libraries can be used to save and load trained machine learning models.

Code:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
HouseDF = pd.read_csv('USA_Housing.csv')
HouseDF.head()
HouseDF=HouseDF.reset_index()
HouseDF.head()
HouseDF.info()
HouseDF.describe()
HouseDF.columns
```

```
sns.pairplot(HouseDF)
sns.distplot(HouseDF['Price’])
sns.heatmap(HouseDF.corr(), annot=True)
X = HouseDF[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Avg. Area
Number of Bedrooms', 'Area Population']]
y = HouseDF['Price’]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.4,random_state=101)
from sklearn.linear_model import minmaxscaler
lm = minmaxscaler(feature_range=(0,1))
lm.fit_transform(X_train,y_train)
print(lm.intercept_)
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient’])
coeff_df
14
from keras.layersimport Dense,Dropout,LSTM
from keras.models import Sequential
model = Sequential()
model.add(LSTM(units = 50,activation = 'relu’, return_sequences = True,input_shape =
(x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units = 60,activation = 'relu’, return_sequences = True))
model.add(Dropout(0.3))
model.add(LSTM(units = 80,activation = 'relu’, return_sequences = True))
model.add(Dropout(0.4))
model.add(LSTM(units = 120,activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 1))
model. Compile(optimizer='adam', loss = 'mean_squared_error’)
model.fit(x_train, y_train,epochs=50)
print(lm.intercept_)
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient’])
coeff_df
predictions = lm.predict(X_test)
scale_factor = 1/0.02099517
y_predicted = y_predicted * scale_factory
```

```
y_test = y_test * scale_factor
plt.scatter(y_test,predictions)
sns.distplot((y_test-predictions),bins=50);
15
plt.figure(figsize=(12,6))
plt.plot(y_test,'b',label = 'Original Price')
plt.plot(y_predicted,'r',label = 'Predicted Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.show()
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```