```
"""
Created on Mon Aug 15 13:12:11 2022


"""

import numpy as np
from fastapi import FastAPI, Form, Request
from fastapi.responses import HTMLResponse
import pandas as pd
from typing import Optional
import numpy as np
from pydantic import BaseModel
from starlette.responses import HTMLResponse
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
# Implementing linear regression
from sklearn.linear_model import LinearRegression
import copy as cp
import joblib

app = FastAPI()

def preprocessing(df1):
  data_copy = df1.copy()

  num_cols = data_copy.select_dtypes(include=['number']).columns.tolist()
  # Combining basement quality features
  features_to_combine1 = ['BsmtQual', 'BsmtCond', 'BsmtExposure']
  mapping1 = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1, "No": 1, "NA": 0, "N/A":
0}
  data_copy[features_to_combine1] =
data_copy[features_to_combine1].replace(mapping1)

  data_copy["BsmtRating"]= data_copy[features_to_combine1].apply(
    lambda x: (x['BsmtQual'] + 5 - x["BsmtCond"]) + x["BsmtExposure"],axis=1)
  data_copy.drop(features_to_combine1, axis=1, inplace=True)

  # Combining basement finish type features
  features_to_combine2 = ['BsmtFinType1', 'BsmtFinType2']
  mapping2 = {"GLQ": 6, "ALQ": 5, "BLQ": 4, "Rec": 3, "LwQ": 2, "Unf": 1, "NA": 0, "N/A": 0}
  data_copy[features_to_combine2] =
data_copy[features_to_combine2].replace(mapping2)
```

```
 data_copy["BsmtFinType"]= data_copy[features_to_combine2].apply(
   lambda x: (x['BsmtFinType1'] + x["BsmtFinType2"]),axis=1)
 data_copy.drop(features_to_combine2, axis=1, inplace=True)

 # Combining garage quality features
 features_to_combine3 = ['GarageQual', 'GarageCond']
 mapping3 = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1, "No": 1, "NA": 0, "N/A":
0}
 data_copy[features_to_combine3] =
data_copy[features_to_combine3].replace(mapping3)

 data_copy["GarageRating"]= data_copy[features_to_combine3].apply(
   lambda x: (x['GarageQual'] + 5 - x["GarageCond"]),axis=1)
 data_copy.drop(features_to_combine3, axis=1, inplace=True)

 # Combining garage type features
 features_to_combine4 = ['GarageType', 'GarageFinish']

 data_copy["GarageInfo"]= data_copy[features_to_combine4].apply(
   lambda x: (x['GarageType'] + "-" + x["GarageFinish"]),axis=1)
 data_copy.drop(features_to_combine4, axis=1, inplace=True)

 cat_cols = data_copy.select_dtypes(include=['object']).columns.tolist()

 # Encode categorical features using LabelEncoder
 encoded_data = cp.deepcopy(data_copy)
 for col in cat_cols:
  encoder_list1 = joblib.load('encoder_list.pkl')
  encoder = encoder_list1[col]
  if(encoder == None):
    mapping = {"Ex": 5, "Gd": 4, "TA": 3, "Av": 3, "Fa": 2, "Mn": 2, "Po": 1, "No": 1, "NA": 0,
"N/A": 0}
    encoded_data[col] = encoded_data[col].replace(mapping)
  else:
    encoded_data[col] = encoder.transform(encoded_data[col])


 X_scaler = joblib.load('X_scaler.pkl')
 num_cols_norm = X_scaler.transform(encoded_data[num_cols])
 num_df = pd.DataFrame(num_cols_norm, columns=num_cols)

 # Replace the original numerical columns with normalized ones
 for col in num_cols:
  encoded_data[col] = num_df[col]
  return encoded_data

features = ['MSZoning',
 'Alley',
 'LotShape',
 'LandContour',
 'LotConfig',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior2nd',
 'MasVnrType',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'PoolQC',
 'Fence',
 'SaleCondition']

feature2 = ['LotFrontage',
 'LotArea',
 'OverallQual',
 'MasVnrArea',
 'BsmtFinSF1',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'LowQualFinSF',
 'GrLivArea',
 'FullBath',
 'HalfBath',
 'TotRmsAbvGrd',
 'Fireplaces',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'ScreenPorch',
 'PoolArea']

# Define default values for each feature
default_values = {
   'MSZoning': 'RL',
   'Alley': 'N/A',
   'LotShape': 'Reg',
   'LandContour': 'Lvl',
   'LotConfig': 'Inside',
   'BldgType': '1Fam',
   'HouseStyle': '1Story',
   'RoofStyle': 'Gable',
   'RoofMatl': 'CompShg',
   'Exterior2nd': 'VinylSd',
   'MasVnrType': 'N/A',
   'ExterQual': 'TA',
   'ExterCond': 'TA',
   'Foundation': 'PConc',
   'BsmtQual': 'TA',
   'BsmtCond': 'TA',
   'BsmtExposure': 'No',
   'BsmtFinType1': 'Unf',
   'BsmtFinType2': 'Unf',
   'Heating': 'GasA',
   'HeatingQC': 'Ex',
   'CentralAir': 'Y',
   'Electrical': 'SBrkr',
   'KitchenQual': 'TA',
   'Functional': 'Typ',
   'FireplaceQu': 'N/A',
   'GarageType': 'Attchd',
   'GarageFinish': 'Unf',
   'GarageQual': 'TA',
   'GarageCond': 'TA',
   'PavedDrive': 'Y',
   'PoolQC': 'N/A',
   'Fence': 'N/A',
   'SaleCondition': 'Normal',



}


class HouseDataCat(BaseModel):
   __annotations__ = {
     feature: Optional[str] for feature in features
   }

class HouseDataNum(BaseModel):
   __annotations__ = {
     feature: Optional[float] for feature in feature2
   }

@app.get('/predict', response_class=HTMLResponse) #data input by forms
# def take_inp():
#   html_content = """
#   <div style="padding-top: 20px; margin-left: 30%;">
#     <h2> House Price Prediction </h2>
#     <form method="post" action="/predict" enctype="application/x-www-form-
urlencoded">
#   """

#   for feature in features:
#     html_content += f"""
#       <div style="margin-bottom: 10px; margin-right:2px; align-right: 2px;">
#         <label for="{feature}">{feature.replace('_', '').capitalize()}:</label>
#         <input type="text" id="{feature}" name="{feature}">
#       </div>
#     """

#   html_content += """
#       <button type="submit">Predict</button>
#     </form>
#   </div>
#   """
#    return HTMLResponse(content=html_content, status_code=200)
def take_inp():
    html_content = """
    <div style="padding-top: 20px; margin-left: 10%; margin-right: 10%;">
        <h2>House Price Prediction</h2>
        <form method="post" action="/predict" enctype="application/x-www-form-
urlencoded">
            <div style="display: flex;">
                <div style="width: 50%; float: left;">
                    <h3>Categorical Features</h3>
    """

    # Add input fields for the first list of features on the left
    for feature in features:
        default_value = default_values.get(feature, "")
        html_content += f"""
            <div style="margin-bottom: 10px;">
                <label for="{feature}">{feature.replace('_', '').capitalize()}:</label>
                <input type="text" id="{feature}" name="{feature}" value="{default_value}">
            </div>
        """

    html_content += """
                </div>
                <div style="width: 50%; float: right;">
                    <h3>Numerical Features</h3>
    """

    # Add input fields for the second list of features on the right
    for feature in feature2:
        html_content += f"""
            <div style="margin-bottom: 10px;">
                <label for="{feature}">{feature.replace('_', '').capitalize()}:</label>
                <input type="number" id="{feature}" name="{feature}" value='0'>
            </div>
        """

    html_content += """
                </div>
            </div>
            <div style="clear: both; text-align: center;">
                <button type="submit">Predict</button>
            </div>
        </form>
    </div>
    """

    return HTMLResponse(content=html_content, status_code=200)



@app.post('/predict')
async def predict(request: Request):

    # Parse the form data
    form_data = await request.form()

    # Initialize a dictionary to store input data
    input_data_cat = {feature: form_data.get(feature) for feature in features}
    input_data_num = {feature: float(form_data.get(feature)) for feature in feature2}

    # Use Pydantic model to validate input and handle missing values
    house_data_cat = HouseDataCat(**input_data_cat)
    house_data_num = HouseDataNum(**input_data_num)
    input_dict1 = house_data_cat.dict()
    input_dict2 = house_data_num.dict()
    input_dict = input_dict1 | input_dict2
    df = pd.DataFrame([input_dict])

    x_test = preprocessing(df)

    model = joblib.load('lin_cv.pkl')
    feature_lst = model.feature_names_in_
    y_scaler = joblib.load("y_scaler.pkl")
    x_test_f = x_test.reindex(columns= feature_lst)
    # x_test_f.replace('N/A',0, inplace=True)
    print(x_test_f)
    prediction = model.predict(x_test_f)

    unscaled_pred = y_scaler.inverse_transform(prediction.reshape(-1, 1))[0][0]
    print("prediction",unscaled_pred)
    # Return the prediction
    return {
        "predicted_price": unscaled_pred # Replace with your actual prediction result
    }

@app.get('/')
```

```
def basic_view():
   return {"<H2>WELCOME to House Price Prediction</H2>": "GO TO /docs route, or /post
or send post request to /predict "}
```