tp2

April 14, 2025

1 TP2 IIA

1.1 Integrantes

- Juan Ignacio García (a2008)
- Rodrigo Mesa Marchi (a2016)
- Danilo Simón Reitano Andrades (a2020)

1.1.1 Tareas y preguntas a resolver:

1. Obtener la correlación entre los atributos y entre los atributos y el target. ¿Qué atributo tiene mayor correlación lineal con el target? ¿Cuáles atributos parecen estar más correlacionados entre sí? Se pueden obtener los valores de correlación o graficarlos directamente utilizando un mapa de calor.
2. Graficar los histogramas de los diferentes atributos y el target. ¿Qué tipo de forma tienen los histogramas? ¿Se observa alguna forma de campana que sugiera que los datos provienen de una distribución gaussiana, sin realizar pruebas de hipótesis?
3. Calcular la regresión lineal utilizando todos los atributos. Con el conjunto de entrenamiento, calcular la varianza total del modelo y la varianza explicada por el modelo. ¿Está el modelo capturando el comportamiento del target? Expanda su respuesta.
4. Calcular las métricas de MSE, MAE y R^2 para el conjunto de evaluación.
5. Crear una regresión de Ridge. Usando validación cruzada de 5 folds y tomando como métrica el MSE, calcular el mejor valor de , buscando entre [0, 12.5]. Graficar el valor de MSE versus .
6. Comparar entre la regresión lineal y la mejor regresión de Ridge los resultados obtenidos en el conjunto de evaluación. ¿Cuál de los dos modelos da mejores resultados (usando MSE y MAE)? Conjeturar por qué el modelo que da mejores resultados mejora. ¿Qué error se puede haber reducido?

Se importan en primer lugar todas las dependencias necesarias de *Scikit Learn*, incluyendo el dataset con el que trabajará: `fetch_california_housing`, que contiene la siguiente información:

- MedInc: Ingreso medio en el bloque
- HouseAge: Edad mediana de las casas en el bloque
- AveRooms: Número promedio de habitaciones por hogar
- AveBedrms: Número promedio de dormitorios por hogar
- Population: Población del bloque
- AveOccup: Número promedio de miembros por hogar
- Latitude: Latitud del bloque
- Longitude: Longitud del bloque

- MedHouseVal: Mediana del costo de las casas en el bloque (en unidades de $100,000). Este es el valor que se intenta predecir

[1]:
```
# Importamos elementos de sklearn
from sklearn.datasets import fetch_california_housing #Dataset
from sklearn.model_selection import train_test_split, cross_val_score #␣
 ↪Division de sets
from sklearn.model_selection import KFold #Kfold
from sklearn.linear_model import LinearRegression # Modelo regresion lineal
from sklearn.linear_model import Ridge, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (r2_score, mean_absolute_error,
                             mean_squared_error, root_mean_squared_error,
                             mean_absolute_percentage_error)
from sklearn.metrics import mean_absolute_error

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

dataset = fetch_california_housing()
```

Se cargan los datos en dos variables diferentes, un dataframe para las variables de entrada, y una serie para las variables de salida; y además realizamos el análisis de sus datos mediante `describe`.

Se agrega el análsis del percentil 99, ya que en algunas graficas posteriores notamos la prescencia de valores anómalos. Como pruebas posteriores a lo sugerido en el TP, se realizarán todos los cálculos nuevamente filtrando estos valores anómalos, para ver si mejoran los resultados generales del modelo. Se observa por ejemplo que el ingreso maximo es de 15, cuando el percentil 99 es de 10, estos casos muestran la prescencia de valores anómalos, que pueden ser candidatos a ser filtrados.

[2]:
```
X = pd.DataFrame(dataset.get("data"), columns= dataset.get("feature_names"))
y = pd.Series(name = dataset.get("target_names")[0], data= dataset.
 ↪get("target"))

X.describe(percentiles=[0.25, 0.5, 0.75, 0.99])
```

[2]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	\
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	
std	1.899822	12.585558	2.474173	0.473911	1132.462122	
min	0.499900	1.000000	0.846154	0.333333	3.000000	
25%	2.563400	18.000000	4.440716	1.006079	787.000000	
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	

```
99%       10.596540     52.000000     10.357033      2.127541   5805.830000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

           AveOccup      Latitude     Longitude
count  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704
std       10.386050      2.135952      2.003532
min        0.692308     32.540000   -124.350000
25%        2.429741     33.930000   -121.800000
50%        2.818116     34.260000   -118.490000
75%        3.282261     37.710000   -118.010000
99%        5.394812     40.626100   -116.290000
max     1243.333333     41.950000   -114.310000
```

[3]:
```
y.describe(percentiles=[0.25, 0.5, 0.75, 0.99])
```

[3]:
```
count    20640.000000
mean         2.068558
std          1.153956
min          0.149990
25%          1.196000
50%          1.797000
75%          2.647250
99%          5.000010
max          5.000010
Name: MedHouseVal, dtype: float64
```

Se observa para el objetivo que existe un gran número de valores iguales a 5, los cuales se notarán facilmente en el análisis de correlación.

Por último, se prepara un dataframe con todos las columnas disponibles para analizar las correlaciones posteriormente

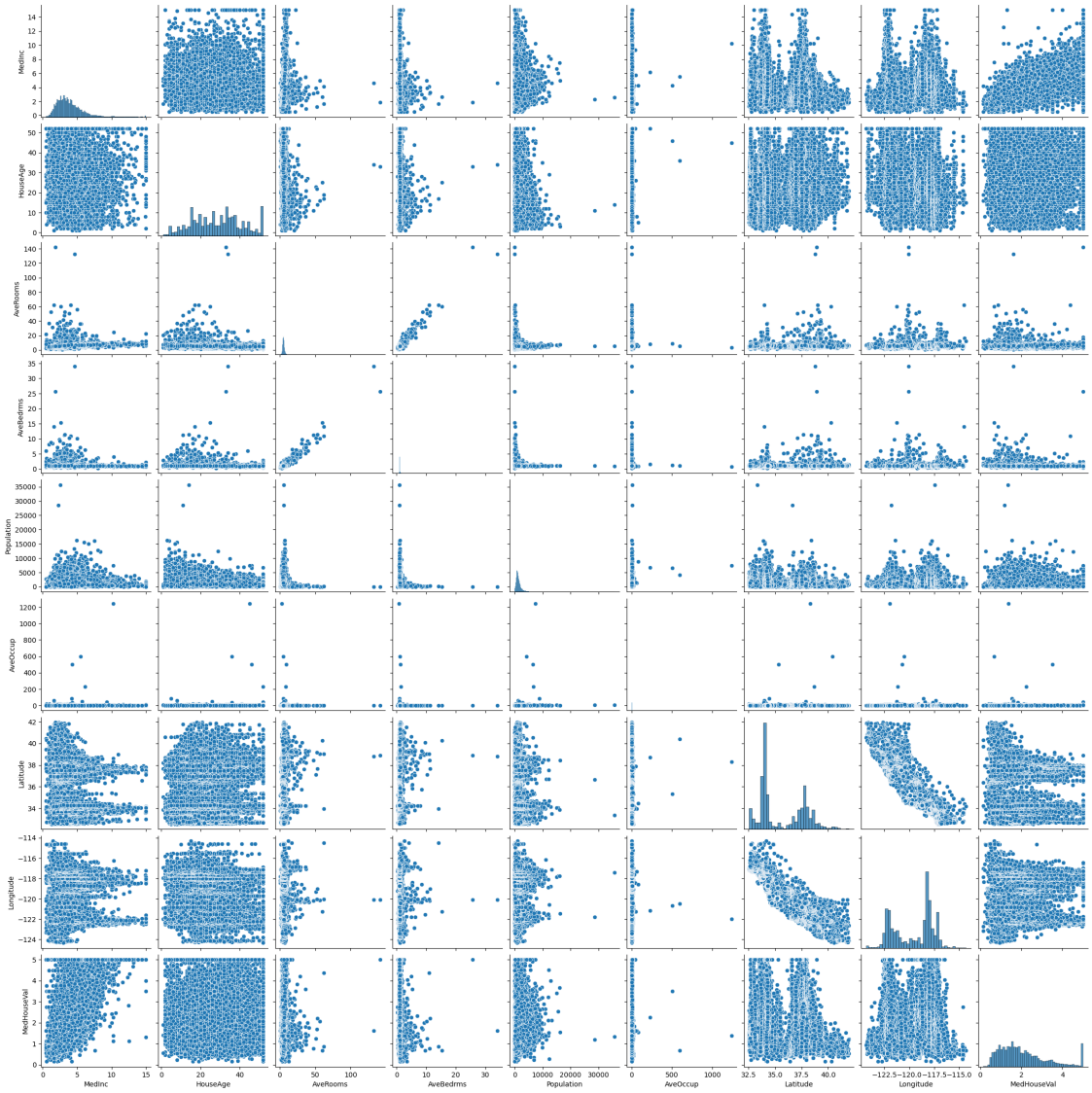
[4]:
```
df = X.copy()
df[y.name] = y
```

1.1.2 Ejercicio 1

En primer lugar se analiza la correlacion entre variables en base a dos graficos específicos: 1. Pairplot 1. Heatmap

[5]:
```
sns.pairplot(df)
```

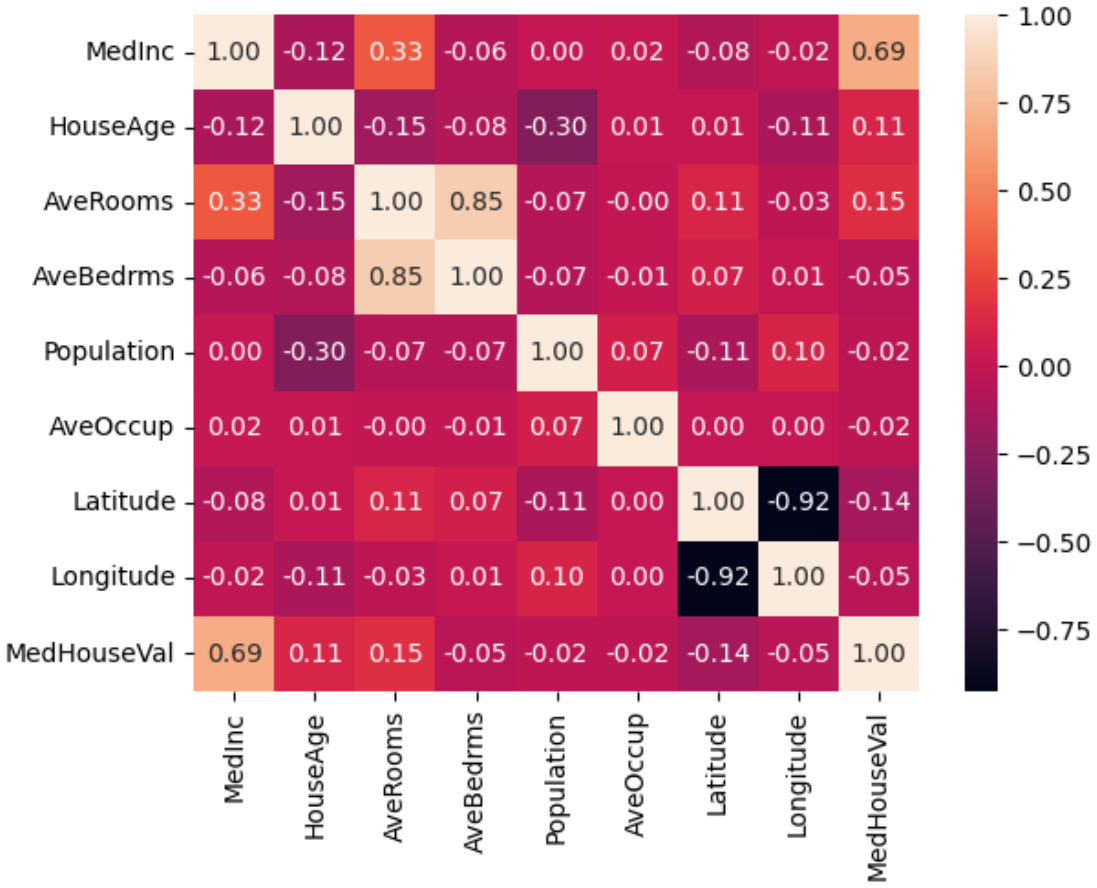
[5]:
```
<seaborn.axisgrid.PairGrid at 0x2906f372e10>
```



Se puede ver que de forma gráfica,la variable que más correlación tiene con la variable de salida `MedHouseVal` es la variable de entrada `MedInc`. También se pueden identificar visualmente algunas correlaciones entre variables de entrada, como entre `AveRooms` y `AveBedrooms` de una casa, y entre `Latitude` y `Longitude`. Esto nos da indicio de que puede existir cierto grado de redundancia entre variables. Para visualizarlo de forma más sencilla se calcula la correlación entre variables:

[6]:
```
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='rocket', annot_kws={"size":
↪10})
```

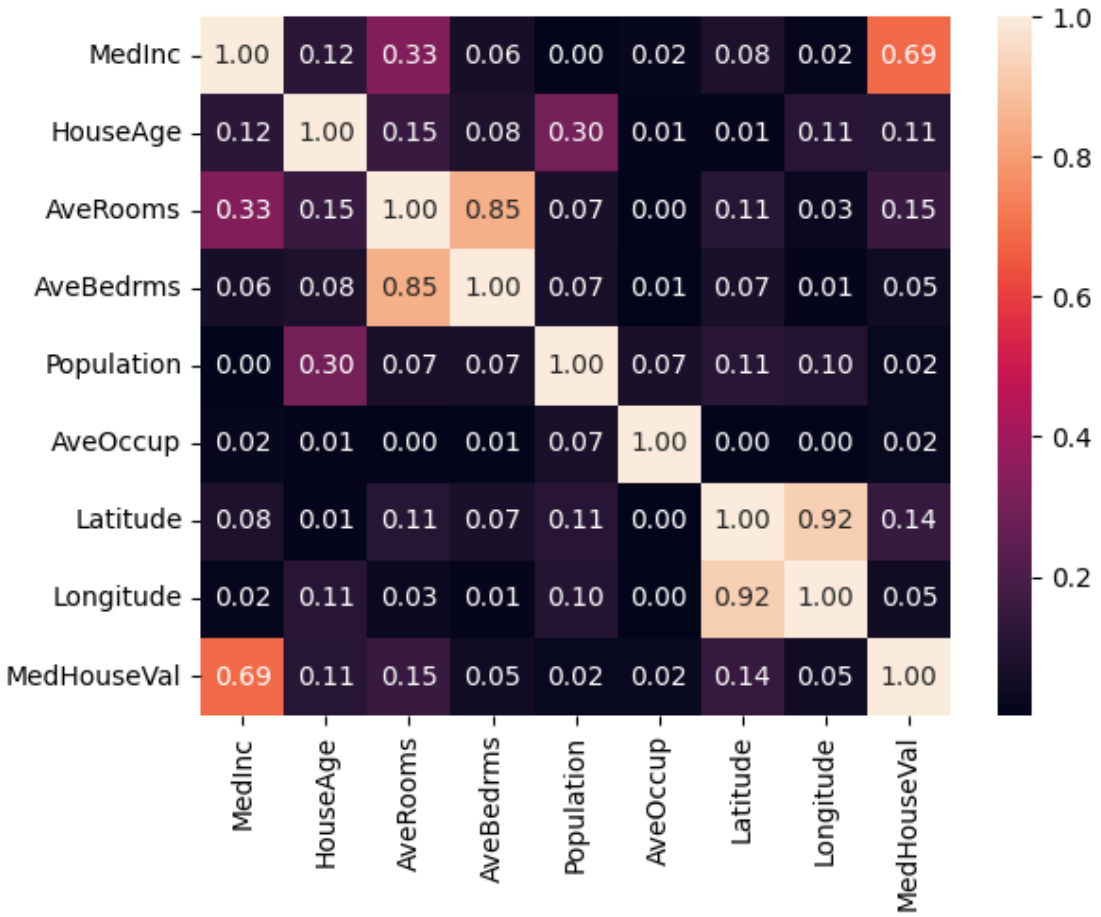
[6]: `<Axes: >`



Se observa a simple vista que se verifica la suposición visual anterior: la variable de entrada que más se correlaciona con la salida es `MedInc`; y entre los features de entrada existe una correlacion entre `AveRooms` y `AveBedrooms`, y entre `Latitude` y `Longitude`. A su vez, para mayor claridad visual, se pueden visualizar los valores absolutos de correlación, dando como resultado:

[7]:
```
sns.heatmap(np.abs(df.corr()), annot=True, fmt='.2f', cmap='rocket',␣
↪annot_kws={"size":10})
```

[7]: `<Axes: >`



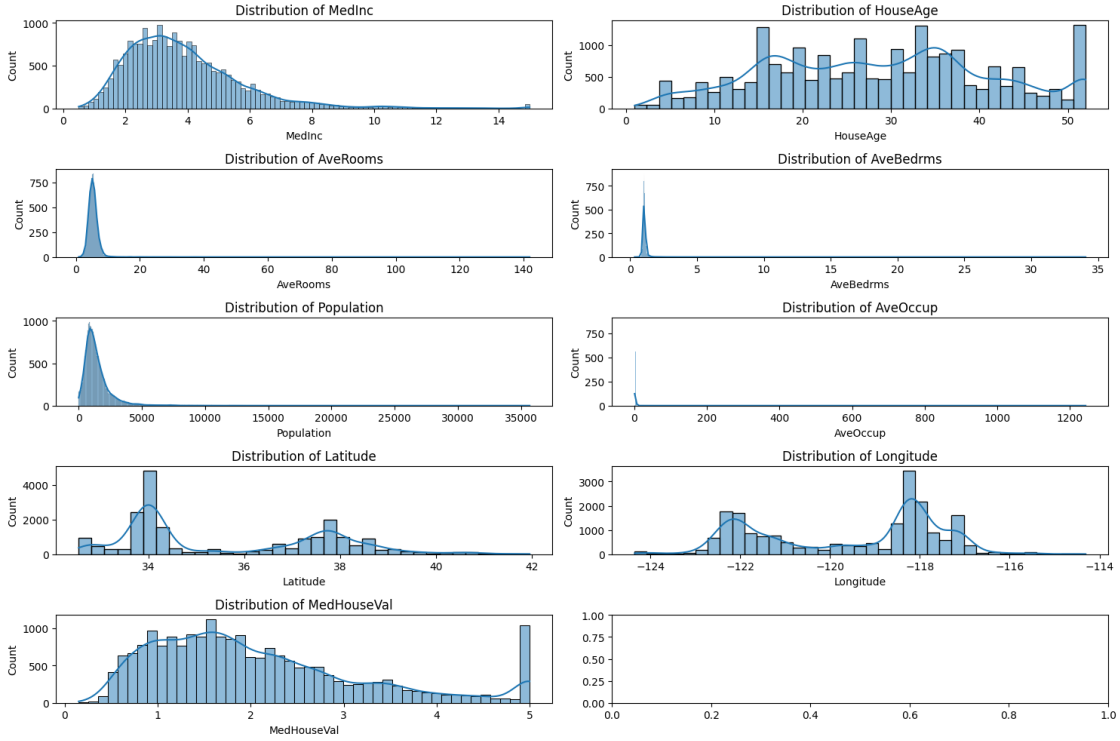
Con esto se comprueba que: 1. `MedInc` es la variable con mayor correlacion con la variable de salida 1. Existen dos correlaciones significativas entre variables de entrada: - `AveRooms` y `AveBedrms` - `Latitude` y `Longitude` 1. Otras correlaciones menos significativas: - `AveRooms` y `MedInc` - `HouseAge` y `Population`

1.1.3 Ejercicio 2:

Ahora se analizan los histogramas que se tienen para los distintos atributos y el target:

[8]:
```
def graph_histograms(df):
    fig, axes = plt.subplots(5, 2, figsize=(15, 10))

    columns = df.columns

    for ax, col in zip(axes.flat, columns):
        sns.histplot(data=df, x=col, kde=True, ax=ax)
        ax.set_title(f'Distribution of {col}')
```

```
    plt.tight_layout()
    plt.show()

graph_histograms(df)
```



Podemos observar lo que se mencionaba con el analisis de los percentiles 99: tenemos varias distribuciones de las variables de entrada que parecen tener valores anómalos: 1. Las variables `AveRooms`, `AveBedrms`, `Population` y `AveOccup` parecen tener valores anómalos, si bien estos datos pueden ser reales, o causa de un error de introducción de la información; estos pueden estar distorsionando los resultados finales, para casos muy esquina. 1. Algunas de las distribuciones de variables, como `AveRooms`, `AveBedrms` y `Population` parecerían tener una distribucion normal, sobre todo si le quitaran los valores anómalos mayores al percentil 99. 1. Para la variable de salida se ve que en general tiene una distribucion similar a la distribucion de `MedInc` (una distribucion similar a una log normal), salvo por ese alto valor acumulado en torno a 5. Esto se manifiesta en la `pairplot` como la línea recta que se encuentra en torno a 5 en todas las gráficas.

1.1.4 Ejercicio 3:

El siguiente paso será analizar la regresión lineal para todos los atributos de entrada y la variable de salida. Para ello en primer lugar se divide el dataset en dos conjuntos: uno que sirva de entrenamiento y otro de evaluación de desempeño del modelo.

[9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.3,␣
↪random_state= 420) # Tomamos un tamaño del set de entrenamiento de 30%
```

Se define también un preprocesador que escala las variables numéricas del dataset para cada columna, que resultan ser todas las variables de entrada disponibles. Una vez creado este módulo, se introduce en un pipeline que será capaz de preparar los datos y pasarlos por el modelo de regresión lineal que se desea utilizar.

[10]:
```
numerical_columns = X.columns

# Creamos el preprocesamiento para las columnas
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_columns)    # Escalamos las␣
 ↪variables numéricas
    ]
)

# Creamos el pipeline con preprocesamiento y modelo
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())  # Aplicamos la regresión lineal
])

pipeline
```

[10]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num', StandardScaler(),
                                                  Index(['MedInc', 'HouseAge',
'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude'],
      dtype='object'))])),
                ('regressor', LinearRegression())])
```

Por último, se entrena el modelo llamando al ajuste del pipeline con los valores de entrenamiento:

[11]:
```
pipeline.fit(X_train, y_train)
```

[11]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num', StandardScaler(),
                                                  Index(['MedInc', 'HouseAge',
'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude'],
      dtype='object'))])),
                ('regressor', LinearRegression())])
```

Tras ajustar el modelo, se pueden observat los valores de ajuste y el efecto que ha causado el preprocesador utilizado:

[12]:
```
print(pipeline.named_steps['preprocessor'].get_feature_names_out())
print(f"El valor de la intersección de la recta es {np.round(pipeline.
 ↪named_steps['regressor'].intercept_, 2)}")
```

```
print(f"Los valores de los coeficientes de la recta son {np.round(pipeline.
 ↪named_steps['regressor'].coef_,2)}")
```

```
['num__MedInc' 'num__HouseAge' 'num__AveRooms' 'num__AveBedrms'
 'num__Population' 'num__AveOccup' 'num__Latitude' 'num__Longitude']
El valor de la intersección de la recta es 2.06
Los valores de los coeficientes de la recta son [ 0.83  0.13 -0.27  0.31  0.
-0.04 -0.88 -0.85]
```

Se calcula el coeficiente R^2 y la desviación estándar del modelo para el conjunto de entrenamiento utilizado:

[13]:
```
print(f"El coeficiente de determinación (R^2) es {pipeline.
 ↪score(X_train,y_train)}")

y_model = pipeline.predict(X_train)
num_attributes = len(pipeline.named_steps['preprocessor'].
 ↪get_feature_names_out())

std_dev_model = np.sqrt((np.sum((y_train - y_model)**2))/(y_train.size -␣
 ↪num_attributes - 1))
print(f"Desvío estándar del modelo {round(std_dev_model, 3)}")
```

```
El coeficiente de determinación (R^2) es 0.6067440341875014
Desvío estándar del modelo 0.723
```

Se observa ver que la desviación estándar del modelo entrenado con respecto a los valores reales de salida es significativo: para un rango de valores de salida entre 0 y 5, una desviación de 0.7 entre valores reales y salidas del modelo es algo muy significativo, indicando que el modelo no está pudiendo describir de forma adecuada la realidad.

1.1.5 Ejercicio 4:

Ahora se analiza el desempeño del modelo para los datos de evaluación usando diversas métricas:

[14]:
```
y_pred = pipeline.predict(X_test)

r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)

print("R-cuadrado en test:", round(r2, 3))
print("Error absoluto medio:", round(mae, 3))
print("Error cuadrático medio:", round(mse, 3))
print("Raíz de error cuadrático medio:", round(rmse, 3))
print(f"Error absoluto porcentual medio: {mape*100:.2f}%")
```

```
R-cuadrado en test: 0.604
Error absoluto medio: 0.531
Error cuadrático medio: 0.531
Raíz de error cuadrático medio: 0.729
Error absoluto porcentual medio: 30.94%
```

Comparando los resultados de R^2 y *rmse* para los sets de entrenamiento y evaluación, se observa que estos se mantienen similares, indicando que no hubo un *overfitting* del modelo en relación a los datos. Algo que se observa es que el score R^2 parece tener un valor bajo, lo cual podría ser señal de que no se puede construir un modelo que se adapte correctamente ante las variables de entrada suministradas, o que el modelo es mas complejo que lo que permite predecir una regresión lineal. Para evaluar en parte la varianza obtenida del modelo y la real, se calcula la desviacion estándar de los datos generales de salida:

[15]:
```
print(f"Desviacion estandar del la variable y_pred: {np.std(y_pred):.2f}")
```

```
Desviacion estandar del la variable y_pred: 0.90
```

[16]:
```
print(f"Desviacion estandar del la variable y: {np.std(y):.2f}")
```

```
Desviacion estandar del la variable y: 1.15
```

Además, se compara el modelo de regresión implementado, contra simplemente estimar por el valor medio de y:

[17]:
```
mean_profit = np.mean(y_train)

y_pred_baseline = np.full_like(y_test, mean_profit)

r2_baseline = r2_score(y_test, y_pred_baseline)
mae_baseline = mean_absolute_error(y_test, y_pred_baseline)
mse_baseline = mean_squared_error(y_test, y_pred_baseline)
rmse_baseline = root_mean_squared_error(y_test, y_pred_baseline)
mape_baseline = mean_absolute_percentage_error(y_test, y_pred_baseline)

print("R-cuadrado en test:", round(r2_baseline, 3))
print("Error absoluto medio:", round(mae_baseline, 3))
print("Error cuadrático medio:", round(mse_baseline, 3))
print("Raiz de error cuadrático medio:", round(rmse_baseline, 3))
print(f"Error absoluto porcentual medio: {mape_baseline*100:.2f}%")
```

```
R-cuadrado en test: -0.0
Error absoluto medio: 0.91
Error cuadrático medio: 1.342
Raiz de error cuadrático medio: 1.159
Error absoluto porcentual medio: 60.71%
```

La desviación estándar del resultado de evaluación tiene una diferencia apreciable con respecto a la desviación estándar de los datos de salida (y) de evaluación. Esto significa que el modelo no está pudiendo explicar de forma satisfactoria los valores de salida en base a las entradas elegidas. Al analizar las correlaciones entre variables de entradas y la variable de salida se tenían en general

correlaciones bajas, lo que puede explicar el desempeño del modelo. Aún así, si se analiza y compara con el desempeño de utilizar el valor medio de y, se observa que el desempeño del modelo de regresión lineal es significativamente mejor.

1.1.6 Ejercicio 5

A continuación se cambia el modelo para realizar una regresión de Ridge con distintos parámetros de α para evaluar si se puede obtener una regresión que mejore los resultados de la regresión lineal. Para ello, se observa cómo evolucionan los coeficientes de Ridge para los distintos valores del parámetro alpha en el intervalo (0 a 12.5).

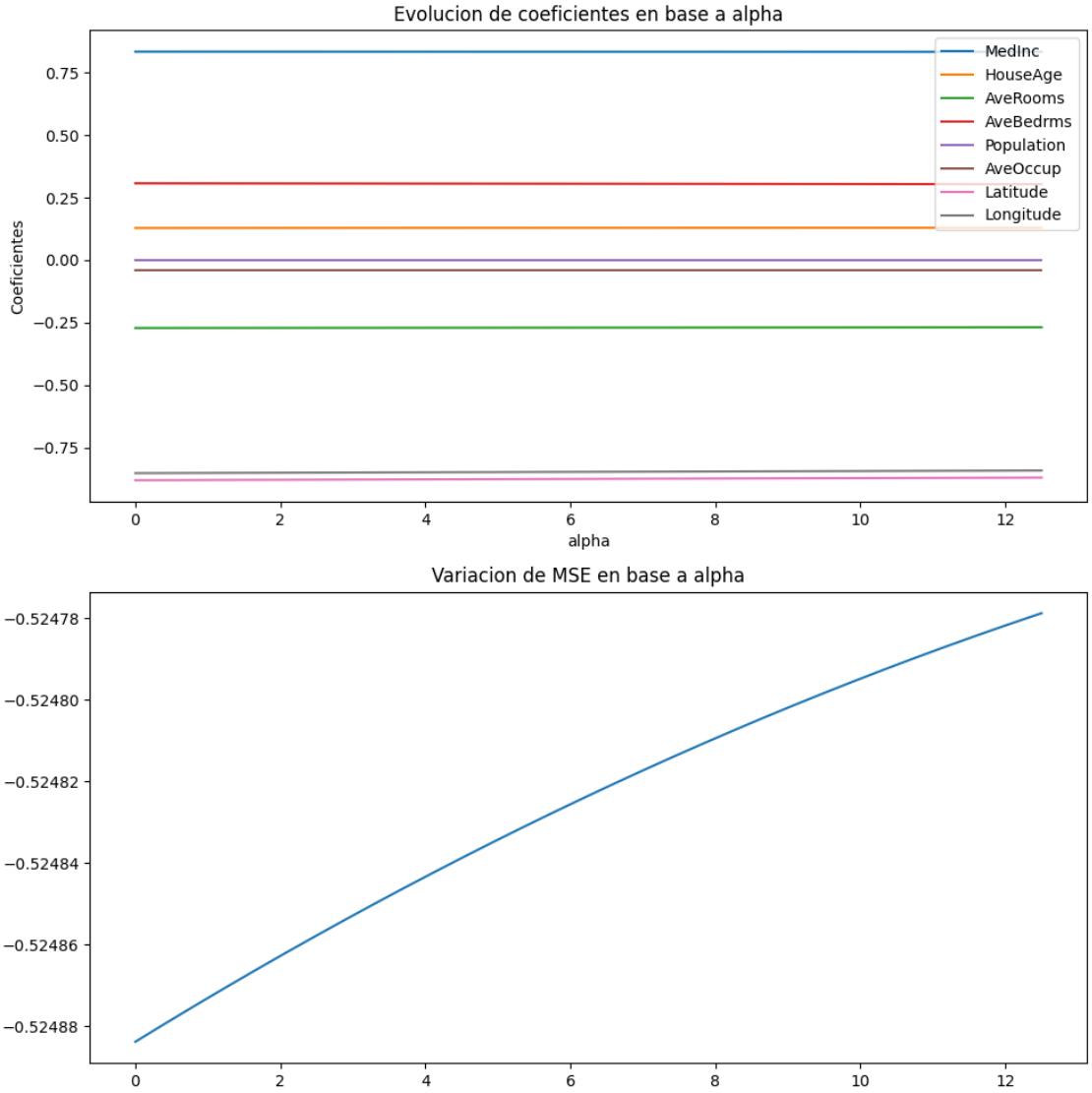
[18]:
```
X_train_processed = preprocessor.fit_transform(X_train)

def show_results(alpha_array:np.ndarray, coeffs_ridge:np.ndarray,␣
 ↪cv_ridge_errors:np.ndarray, best_alpha_ridge:float):

    print(f"El mejor valor de alpha para Ridge es: {best_alpha_ridge}")

    f, ax = plt.subplots(2, 1, figsize = (10,10))
    for i in range(coeffs_ridge.shape[1]):
        ax[0].plot(alpha_array, coeffs_ridge[:, i], label=X_train.columns[i])

    ax[0].legend()
    ax[0].set_ylabel("Coeficientes")
    ax[0].set_xlabel("alpha")
    ax[0].set_xlabel("alpha")
    ax[0].set_title("Evolucion de coeficientes en base a alpha")

    ax[1].plot(alpha_array,cv_ridge_errors)
    ax[1].ticklabel_format(axis='y', style='plain')
    ax[1].set_title("Variacion de MSE en base a alpha")
    plt.tight_layout()

def func_evaluate_ridge(alpha_array:np.ndarray, num_folds:int):

    coeffs_ridge = np.zeros([alpha_array.size, 8])

    # Calculamos los coeficientes para diferentes valores de lambda
    for index, alpha in enumerate(alpha_array):

        # Creamos los modelos
        ridge_model = Ridge(alpha=alpha)

        # Si alpha es cero, significa que es una regresión lineal
        if index == 0:
            ridge_model = LinearRegression()

        # Los entrenamos
```

```
        ridge_model.fit(X_train_processed, y_train)

        # Guardamos los coeficientes de las regresiones
        coeffs_ridge[index, :] = ridge_model.coef_.copy()

    # Validación cruzada de k folds
    kf = KFold(n_splits=num_folds, shuffle=True, random_state=420)

    # Donde vamos a almacenar los errores
    cv_ridge_errors = []

    # Búsqueda del mejor alpha usando un loop
    for alpha in alpha_array:
        fold_ridge_errors = []
        ridge_model = Ridge(alpha=alpha)
        # Realizar la validación cruzada
        cv = cross_val_score(ridge_model, X_train_processed, y=y_train,␣
↪scoring="neg_mean_squared_error", cv=num_folds, n_jobs=-1)
        cv_ridge_errors.append(np.mean(cv))



    # Mostramos cual es el mejor valor de alpha para cada caso
    best_alpha_ridge = alpha_array[np.argmax(cv_ridge_errors)]

    show_results(alpha_array, coeffs_ridge,cv_ridge_errors, best_alpha_ridge)
    return best_alpha_ridge
```

Usando validación cruzada con 5 folds, obtenemos los siguientes resultados:

[19]:
```
alpha_array = np.linspace(0, 12.5, 1000)
num_folds = 5
best_alpha_ridge = func_evaluate_ridge(alpha_array, num_folds)
```

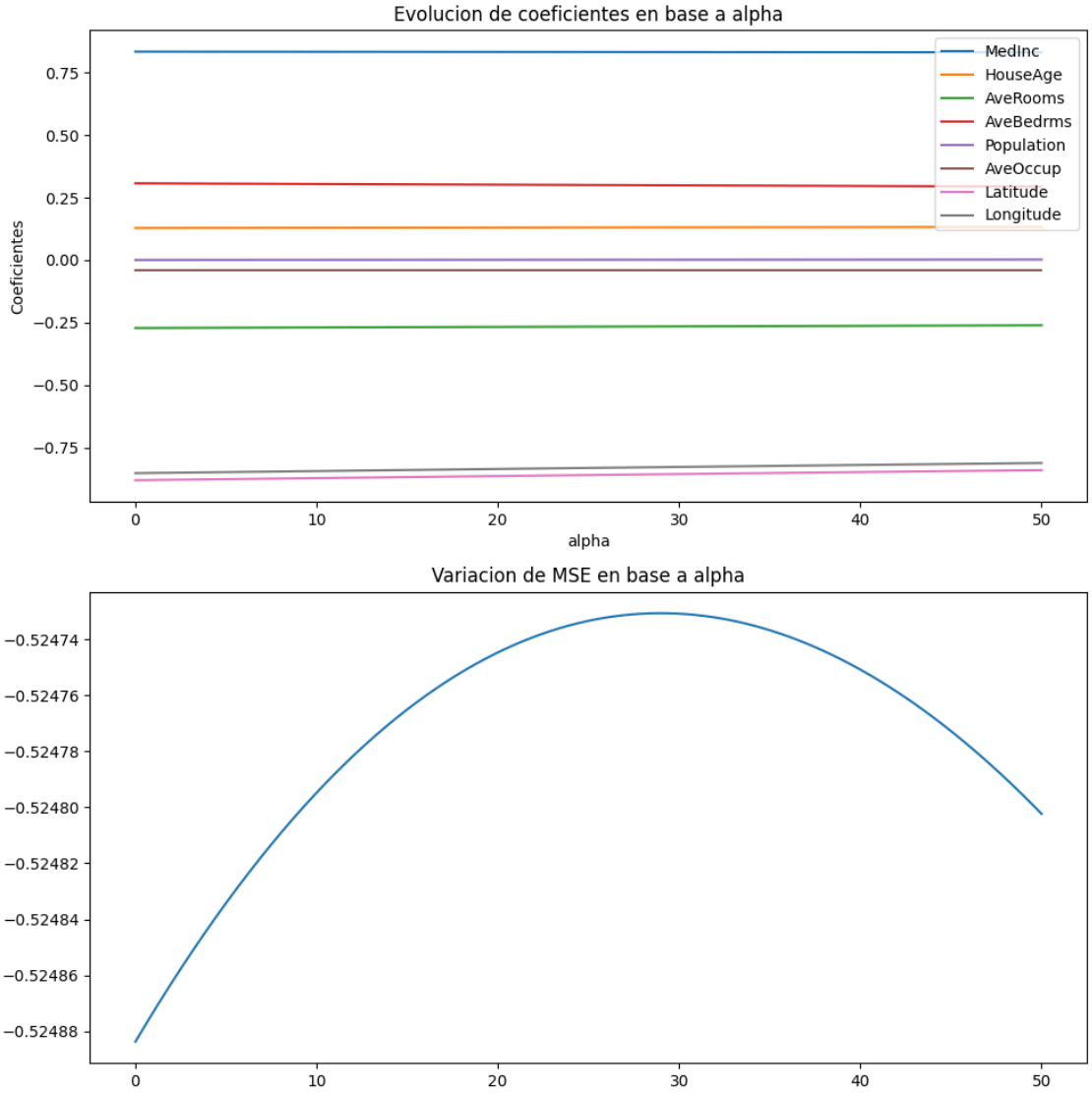
```
El mejor valor de alpha para Ridge es: 12.5
```



Se encuentra que el valor de alfa que mejora la prediccion utilizando 5 folds es $\alpha = 12.5$. Si se interpreta la grafica de la evolución de los coeficientes en base a alpha, se obtienen prácticamente rectas, lo cual indica que se puede estar analizando una porción muy pequeña de los posibles valores de alpha. A su vez, teniendo en cuenta que el mejor valor encontrado de alpha es el límite superior, se intuye que una buena alternativa será evaluar valores de alpha mayores al límite establecido. Si se grafica la variación de MSE al variar alpha en estos rangos no varía de forma significativa, lo cual también muestra que se puede mejorar el valor de alfa obtenido.

[20]:
```
alpha_array = np.linspace(0, 50, 1000)
num_folds = 5
best_alpha_ridge = func_evaluate_ridge(alpha_array, num_folds)
```

```
El mejor valor de alpha para Ridge es: 28.97897897897898
```



Probando estos parámetros, se observa que el modelo mejora hasta aproximadamente un alfa de 28.978. Este valor se utilizará posteriormente en las comparaciones finales.

1.1.7 Ejercicio 6:

Con todos los resultados anteriores, se calculan los MSE y MAE de tres casos posibles: 1. Regresion Lineal 1. Ridge con alpha = 12.5 (rangos propuestos) 1. Ridge con alpha = 28.978 (nuevo valor propuesto)

[21]:
```
def validate_regressor(regressor, preprocessor, X_train, X_test, y_train,␣
↪y_test):
    # Creamos el pipeline con preprocesamiento y modelo
    pipeline = Pipeline(steps=[
```

```
        ('preprocessor', preprocessor),
        ('regressor', regressor)  # Aplicamos la regresión lineal
    ])

    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    return mae, mse
```

[22]:
```
name_list = ["Regresion Lineal", "Ridge alpha=12.5", "Ridge alpha=28.978"]
reg_list = [LinearRegression(), Ridge(alpha=12.5), Ridge(alpha=28.978)]

# Creamos el preprocesamiento para las columnas
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_columns)    # Escalamos las␣
 ↪variables numéricas
    ]
)

for name, reg in zip(name_list, reg_list):
    mae, mse = validate_regressor(reg, preprocessor, X_train, X_test, y_train,␣
 ↪y_test)
    print(f"Valores obtenidos para {name}: MAE = {mae}  MSE = {mse}")
```

```
Valores obtenidos para Regresion Lineal: MAE = 0.5307069814636155  MSE =
0.5309012639324572
Valores obtenidos para Ridge alpha=12.5: MAE = 0.5307347939133844  MSE =
0.5310848046220198
Valores obtenidos para Ridge alpha=28.978: MAE = 0.5307945203589521  MSE =
0.5313698535509757
```

Se concluye que el mejor modelo establecido es el modelo de regresión lineal simple, probablemente debido a que se utiliza un dataset con gran cantidad de datos y que no posee mucha correlación entre variables en líneas generales.

1.1.8 Ajuste previo de los datos

En esta sección se repiten los pasos anteriores filtrando el dataset para eliminar las anomalías detectadas en las columnas `AveRooms`, `AveBedrms`, `Population` y `AveOccup`, recortando para ello las filas que contienen valores que superan el valor dado por el percentil 99 de la distribución de cada columna. El nuevo conjunto de datos queda conformado así:

[23]:
```
columns = ['AveRooms', 'AveBedrms', 'Population', 'AveOccup']
for col in columns:
    print(f"{col}: P99 = {df[col].quantile(0.99):.2f}, Max = {df[col].max():.
 ↪2f}")
```

```
# Filtrar outliers: eliminamos todo lo que esté por encima del P99
df_cleaned = df.copy()
for col in columns:
    p99 = df_cleaned[col].quantile(0.99)
    df_cleaned = df_cleaned[df_cleaned[col] <= p99]

# Verificación
print("Datos después de remover outliers:", df_cleaned.shape)
df_cleaned.describe(percentiles=[0.25, 0.5, 0.75, 0.99])
```

```
AveRooms: P99 = 10.36, Max = 141.91
AveBedrms: P99 = 2.13, Max = 34.07
Population: P99 = 5805.83, Max = 35682.00
AveOccup: P99 = 5.39, Max = 1243.33
Datos después de remover outliers: (19824, 9)
```

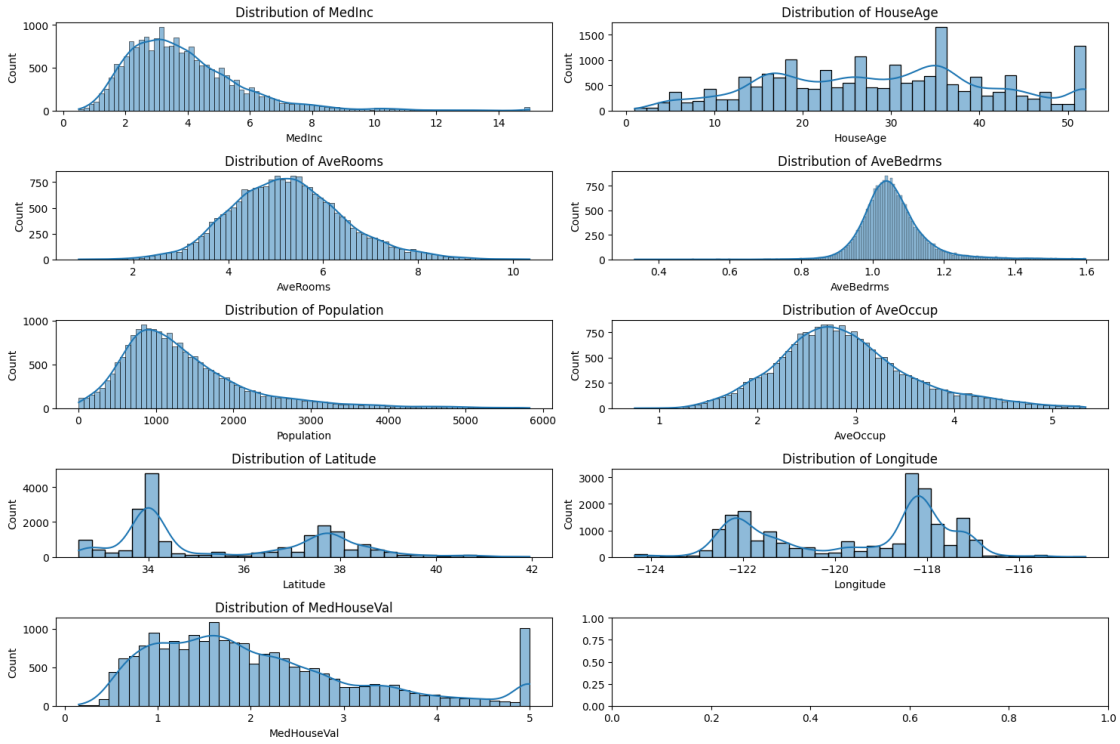
[23]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	\
count	19824.000000	19824.000000	19824.000000	19824.000000	19824.000000	
mean	3.879798	28.962520	5.256983	1.058270	1370.569613	
std	1.895074	12.496172	1.200756	0.094777	864.423601	
min	0.499900	1.000000	0.846154	0.333333	3.000000	
25%	2.566775	19.000000	4.431941	1.004630	798.000000	
50%	3.543900	29.000000	5.201663	1.047015	1168.000000	
75%	4.762625	37.000000	5.991270	1.095276	1705.250000	
99%	10.580994	52.000000	8.405668	1.428571	4575.310000	
max	15.000100	52.000000	10.352941	1.597510	5826.000000	

	AveOccup	Latitude	Longitude	MedHouseVal
count	19824.000000	19824.000000	19824.000000	19824.000000
mean	2.894755	35.621978	-119.588542	2.082108
std	0.692928	2.124274	1.996995	1.157404
min	0.750000	32.540000	-124.350000	0.149990
25%	2.431636	33.930000	-121.810000	1.203750
50%	2.817415	34.250000	-118.500000	1.815500
75%	3.270292	37.710000	-118.020000	2.667000
99%	4.856792	40.610000	-116.450000	5.000010
max	5.337329	41.950000	-114.550000	5.000010

Habiendo eliminado los valores anómalos, se grafica también los histogramas para cada feature, observando que ahora se aprecia mejor la distribución normal que siguen las columnas afectadas.

[24]:
```
graph_histograms(df_cleaned)
```



Por último, se vuelven a obtener los valores de error de los tres modelos, observando una mejora en el desempeño del modelo:

[25]:
```
y_cleaned = df_cleaned["MedHouseVal"]
X_cleaned = df_cleaned.drop(columns=["MedHouseVal"])

X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned =␣
 ↪train_test_split(X_cleaned, y_cleaned, test_size= 0.3, random_state= 420)

name_list = ["Regresion Lineal (cleaned)", "Ridge alpha=12.5 (cleaned)", "Ridge␣
 ↪alpha=28.978 (cleaned)"]
reg_list = [LinearRegression(), Ridge(alpha=12.5), Ridge(alpha=28.978)]

for name, reg in zip(name_list, reg_list):
    mae, mse = validate_regressor(reg, preprocessor, X_train_cleaned,␣
 ↪X_test_cleaned, y_train_cleaned, y_test_cleaned)
    print(f"Valores obtenidos para {name}: MAE = {mae}  MSE = {mse}")
```

```
Valores obtenidos para Regresion Lineal (cleaned): MAE = 0.49129889313345115
MSE = 0.4438375137418894
Valores obtenidos para Ridge alpha=12.5 (cleaned): MAE = 0.4912464239843738  MSE
= 0.443902798329757
Valores obtenidos para Ridge alpha=28.978 (cleaned): MAE = 0.4912405250631114
MSE = 0.444042804210961
```

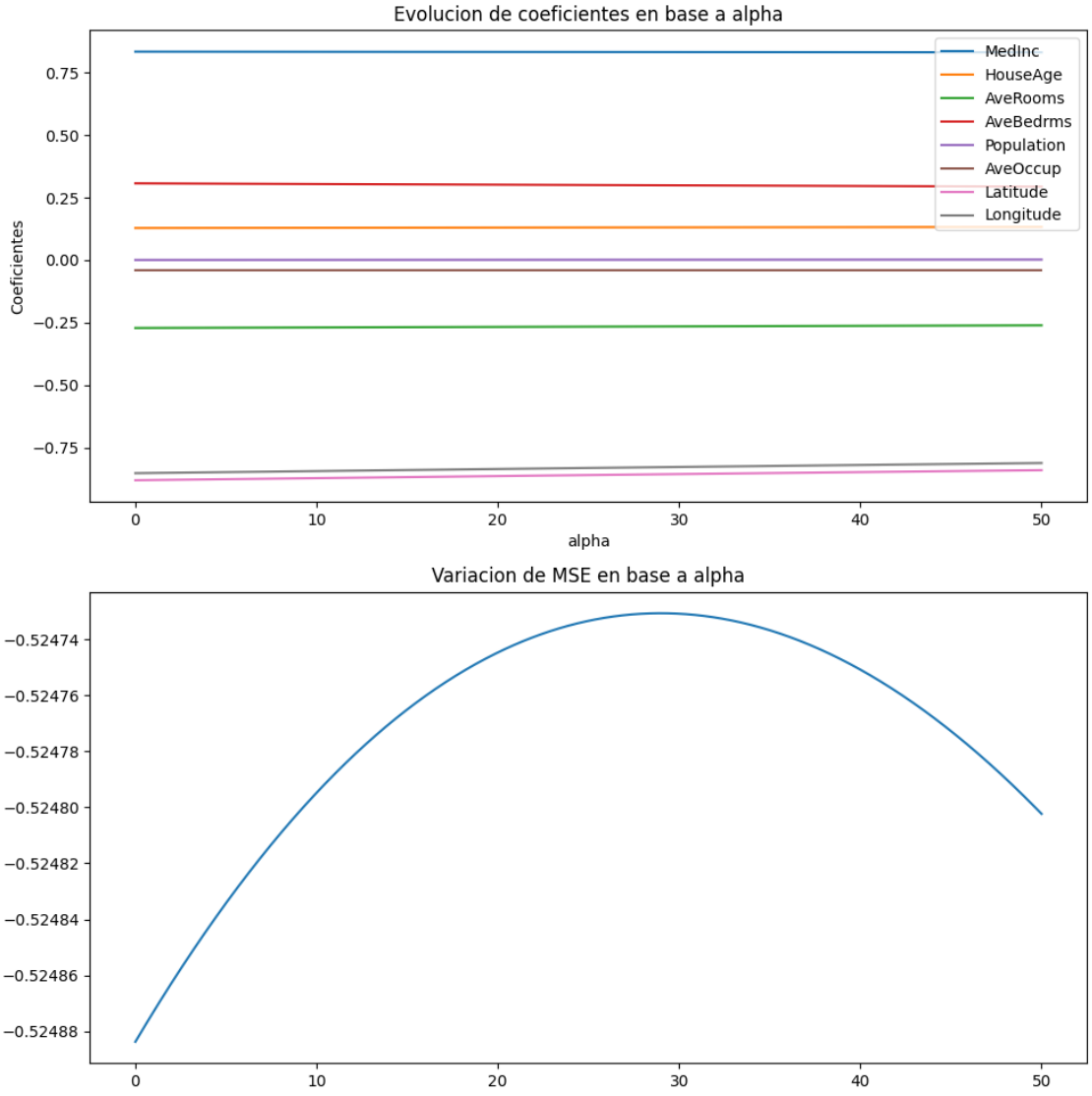
1.1.9 Regularización L1

Adicionalmente se plantea aplicar una regularización de Lasso a modo comparativo:

[27]:
```
def show_results(alpha_array:np.ndarray, coeffs_lasso:np.ndarray,␣
↪cv_lasso_errors:np.ndarray, best_alpha_lasso:float):
    print(f"El mejor valor de alpha para Lasso es: {best_alpha_lasso}")

    f, ax = plt.subplots(2, 1, figsize = (10,10))
    for i in range(coeffs_lasso.shape[1]):
        ax[0].plot(alpha_array, coeffs_lasso[:, i], label=X_train.columns[i])

    ax[0].legend()
    ax[0].set_ylabel("Coeficientes")
    ax[0].set_xlabel("alpha")
    ax[0].set_xlabel("alpha")
    ax[0].set_title("Evolucion de coeficientes en base a alpha")

    ax[1].plot(alpha_array,cv_lasso_errors)
    ax[1].ticklabel_format(axis='y', style='plain')
    ax[1].set_title("Variacion de MSE en base a alpha")
    plt.tight_layout()

def func_evaluate_lasso(alpha_array:np.ndarray, num_folds:int):

    coeffs_lasso = np.zeros([alpha_array.size, 8])

    # Calculamos los coeficientes para diferentes valores de lambda
    for index, alpha in enumerate(alpha_array):

        # Creamos los modelos
        lasso_model = Lasso(alpha=alpha)

        # Si alpha es cero, significa que es una regresión lineal
        if index == 0:
            lasso_model = LinearRegression()

        # Los entrenamos
        lasso_model.fit(X_train_processed, y_train)

        # Guardamos los coeficientes de las regresiones
        coeffs_lasso[index, :] = lasso_model.coef_.copy()

    # Validación cruzada de k folds
    kf = KFold(n_splits=num_folds, shuffle=True, random_state=420)

    # Donde vamos a almacenar los errores
    cv_lasso_errors = []
```

```
    # Búsqueda del mejor alpha usando un loop
    for alpha in alpha_array:
        fold_lasso_errors = []
        lasso_model = Lasso(alpha=alpha)
        # Realizar la validación cruzada
        cv = cross_val_score(lasso_model, X_train_processed, y=y_train,␣
 ↪scoring="neg_mean_squared_error", cv=num_folds, n_jobs=-1)
        cv_lasso_errors.append(np.mean(cv))

    # Mostramos cual es el mejor valor de alpha para cada caso
    best_alpha_lasso = alpha_array[np.argmax(cv_lasso_errors)]

    show_results(alpha_array, coeffs_lasso,cv_lasso_errors, best_alpha_lasso)
    return best_alpha_lasso

alpha_array = np.linspace(0, 50, 1000)
num_folds = 5
best_alpha_ridge = func_evaluate_ridge(alpha_array, num_folds)

mae, mse = validate_regressor(Ridge(alpha=28.978), preprocessor,␣
 ↪X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned)
print(f"Valores obtenidos para Lasso (cleaned): MAE = {mae}  MSE = {mse}")
```

```
El mejor valor de alpha para Lasso es: 28.97897897897898
Valores obtenidos para Lasso (cleaned): MAE = 0.4912405250631114  MSE =
0.444042804210961
```



Se observa que para este conjunto de datos no existe una mejora en el desempeño con respecto a las técnicas utilizadas. Aparentemente, para este conjunto de datos modificado, el modelo que mejor estima las muestras es un **modelo lineal al que no se le aplica regularización**.