UNIVERSIDAD DE BUENOS AIRES

RECONOCIMIENTO DE PATRONES

Trabajo Práctico N°1

Evelyn Olszanowski
Ana Carolina Heidenreich
Bruno Gomez

supervisado por

Dr. Julio JACOBO

May 18, 2021

1 Introducción

La regresión lineal es util para modelar relaciones entre la variable que se desea predecir y el resto de covariables. Los modelos que emplean parámetros de regularización atenúan problemas relacionados con la presencia de predictores correlacionados, riesgo de *overfitting* e inclusion de predictores poco relevantes, mediante la penalizacion de los coeficientes del modelo (w), como se muestra en la ecuación (1). Tiene el efecto de reducir de forma proporcional el valor de todos los coeficientes del modelo pero sin que estos lleguen a cero. El grado de penalización está controlado por el hiperparámetro λ. Cuando $\lambda = 0$, la penalización es nula y el resultado es equivalente al de un modelo lineal por mínimos cuadrados ordinarios (OLS). A medida que λ aumenta, mayor es la penalización y menor el valor de los predictores.

$$\sum_{n=1}^{N}(t_n - \boldsymbol{w^t}\boldsymbol{\phi}(x_n))^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q = \text{suma residuos cuadrados} + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q \tag{1}$$

Las estrategias más conocidas que incorporan este tipo de penalización son:

1.1 Ridge

La regularización Ridge penaliza la suma de los coeficientes elevados al cuadrado ($||w||_2^2 = \sum_{j=1}^{M} w_j^2$). Para el caso de Ridge el valor de q en la ecuación (1) es dos.

1.2 LASSO

La regularización Lasso penaliza la suma de los valores absolutos de los coeficientes de regresión ($||w||_1 = \sum_{j=1}^{M}|w_j|$). Para el caso de Lasso el valor de q en la ecuación (1) es uno.

1.3 Elastic net

Elastic net incluye una regularización que combina la penalización de norma 1 y norma 2 a través del parámetro r, el cual controla el grado en que influye cada una de las penalizaciones. Su valor está comprendido entre 0 y 1. Cuando r=0, se aplica Ridge y cuando r=1 se aplica LASSO. La combinación de ambas penalizaciones suele dar lugar a buenos resultados. Una estrategia frecuentemente utilizada es asignarle casi todo el peso a la penalización con norma 1 (r muy próximo a 1) para conseguir seleccionar predictores y un poco a la penalización con norma 2 para dar cierta estabilidad en el caso de que algunos predictores estén correlacionados.

$$(\lambda(r||w||_1 + \frac{1}{2}(1-r)||w||_2^2))$$

1.4 Objetivo

En este trabajo se empleará la base de datos de *California Housing Prices*, que corresponde a un censo del año 1990 en la provincia de California, Estados Unidos. Cada entrada es un distrito caracterizado por diez variables. El objetivo de este trabajo práctico fue predecir el valor de la variable *median_housing_price* utilizando el resto de las covariables del dataset. La estimación se realizó utilizando los tres modelos de regresión con regularización presentados.

[1]:
```
# Importamos los paquetes necesarios
# Python > 3.5
import sys
assert sys.version_info >= (3, 5)

import numpy as np
import os
import seaborn as sns
import pandas as pd

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
from pandas.plotting import scatter_matrix

# Scikit-Learn >0.20
import sklearn
assert sklearn.__version__ >= "0.20"


from sklearn.model_selection import StratifiedShuffleSplit
import matplotlib.pyplot as plt
from zlib import crc32
import hashlib
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from six.moves import urllib
from scipy import stats
from sklearn.metrics import mean_squared_error
from sklearn import metrics
from sklearn import linear_model
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.model_selection import (TimeSeriesSplit, KFold, ShuffleSplit,
StratifiedKFold, GroupShuffleSplit,
```

```
GroupKFold, StratifiedShuffleSplit,cross_val_score)
from sklearn.linear_model import Ridge
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import (OrdinalEncoder,OneHotEncoder)

# Donde guardar las figuras
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "end_to_end_project"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Cargamos los datos y miramos las 5 primeras entradas

[2]:
```
housing = pd.read_csv("data/housing.csv");
housing.head();
```

2 Exploración de los datos

La base de datos cuenta con 10 variables, donde 9 seran features predictoras y "median_house_value" será la variable target.

[3]:
```
housing.head()
```

[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
0	-122.23	37.88	41.0	880.0	129.0	
1	-122.22	37.86	21.0	7099.0	1106.0	
2	-122.24	37.85	52.0	1467.0	190.0	
3	-122.25	37.85	52.0	1274.0	235.0	
4	-122.25	37.85	52.0	1627.0	280.0	

	population	households	median_income	median_house_value	ocean_proximity
0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	496.0	177.0	7.2574	352100.0	NEAR BAY
3	558.0	219.0	5.6431	341300.0	NEAR BAY
4	565.0	259.0	3.8462	342200.0	NEAR BAY

[4]:
```
housing.columns
```

[4]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity'],
      dtype='object')
```

Con gráficos de violin (Figura 1) observamos la distribución de las features. Observamos que 'longitude' y 'latitude' presentan una distribución bimodal. A su vez observamos que 'total_rooms', 'total_bedrooms', 'population', 'households', están sesgadas a derecha y 'median_income', 'median_house_value' tienen un grado menor de sesgo también a derecha.

[5]:
```
colnames = housing.columns
fig = plt.figure(figsize=(16,12))
for i in range(1, 10):
    ax = plt.subplot(3, 3, i)
    ax.set_xlabel(colnames[i-1])
    sns.violinplot(data=housing.iloc[:,i-1], orient= "h")
```

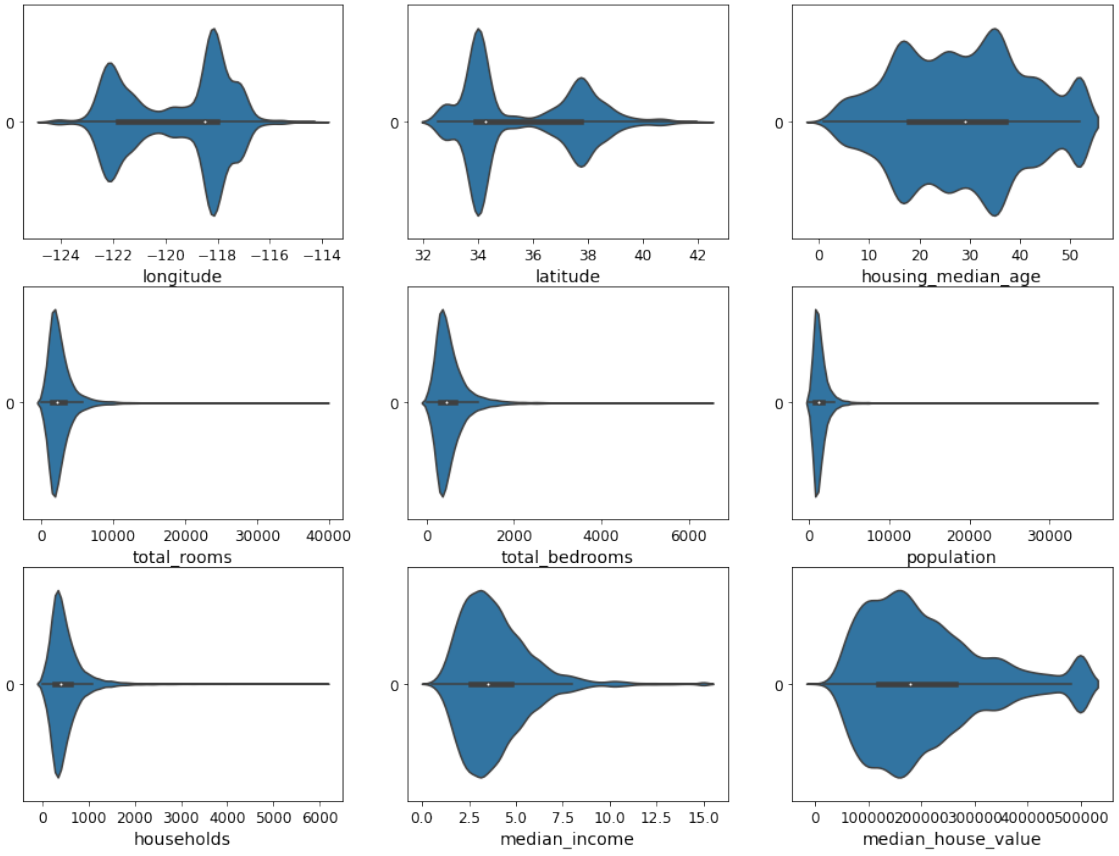

Figura 1

La tabla cuenta con datos de 20640 distritos. Como se puede obsevar en la tabla que sigue, la variable 'total_bedrooms' posee 207 datos faltantes.

[6]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Discretizamos la variable median_income en 5 categorías. Vemos que las proporciones categorías no son uniformes.

[7]:
```
# Download the California image
images_path = os.path.join(PROJECT_ROOT_DIR, "images", "end_to_end_project")
os.makedirs(images_path, exist_ok=True)
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
filename = "california.png"
print("Downloading", filename)
url = DOWNLOAD_ROOT + "images/end_to_end_project/" + filename
urllib.request.urlretrieve(url, os.path.join(images_path, filename))
```

```
Downloading california.png
```

[7]:
```
('./images/end_to_end_project/california.png',
 <http.client.HTTPMessage at 0x7fe3206ce250>)
```

El grafico de la Figura 2 presenta la distribución geográfica de la mediana del valor de las casas por distrito en California. Donde el tamaño de punto es proporcional al tamaño de la población en ese distrito y el color del punto indica el valor de la variable target median_house_value. Se puede observar que las regiones de San Francisco y Los Angeles concentran la mayor cantidad de puntos rojos correspondientes a valores de mediana más elevados por distrito.

[8]:
```
california_img=mpimg.imread(os.path.join(images_path, filename))
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                  s=housing['population']/100, label="Population",
                  c="median_house_value", cmap=plt.get_cmap("jet"),
                  colorbar=False, alpha=0.4)
```

```
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cbar = plt.colorbar(ticks=tick_values/prices.max())
cbar.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values],␣
 ↪fontsize=14)
cbar.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
save_fig("california_housing_prices_plot")
plt.show()
```

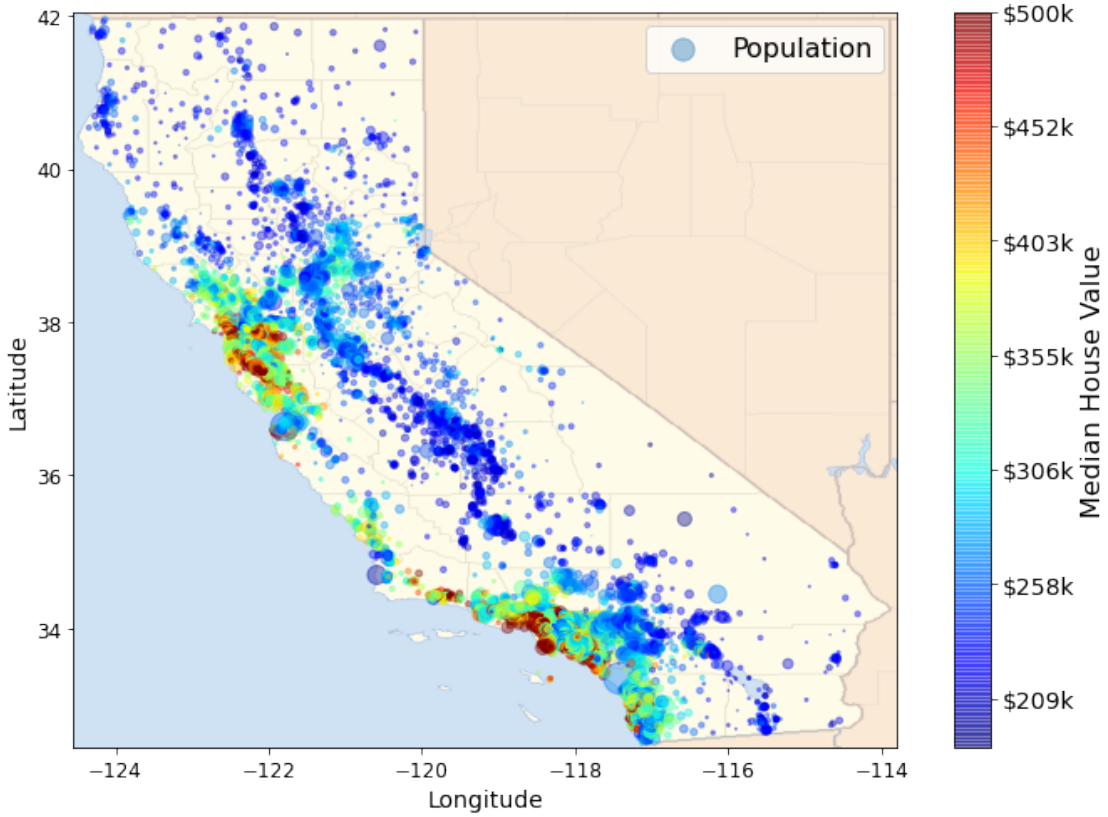

Figura 2

En la Figura 3 se pueden observar los valores de la matriz de correlación entre las variables. Vemos que las variables 'total_rooms', 'total_bedrooms', 'population' y 'households', presentan valores de corelación cercanos a uno, por tanto dichas covariables seran transformadas de manera de resumir la informacion concentrada en ellas tal como se verá en la sección XXX.

[9]:
```
# Correlación entre variables preprocesamiento
corr_matrix = housing.corr()

paleta = sns.diverging_palette(150, 275, s=80, l=40, n=20)

with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(17, 12))
    ax = sns.heatmap(corr_matrix,
                     annot=True,
                     annot_kws={'size': 12},
                     fmt='.2f',
                     vmax=1,
                     vmin=-1,
                     square=True,
                     linewidths=.01,
                     linecolor='lightgray',
                     cmap=paleta)
```

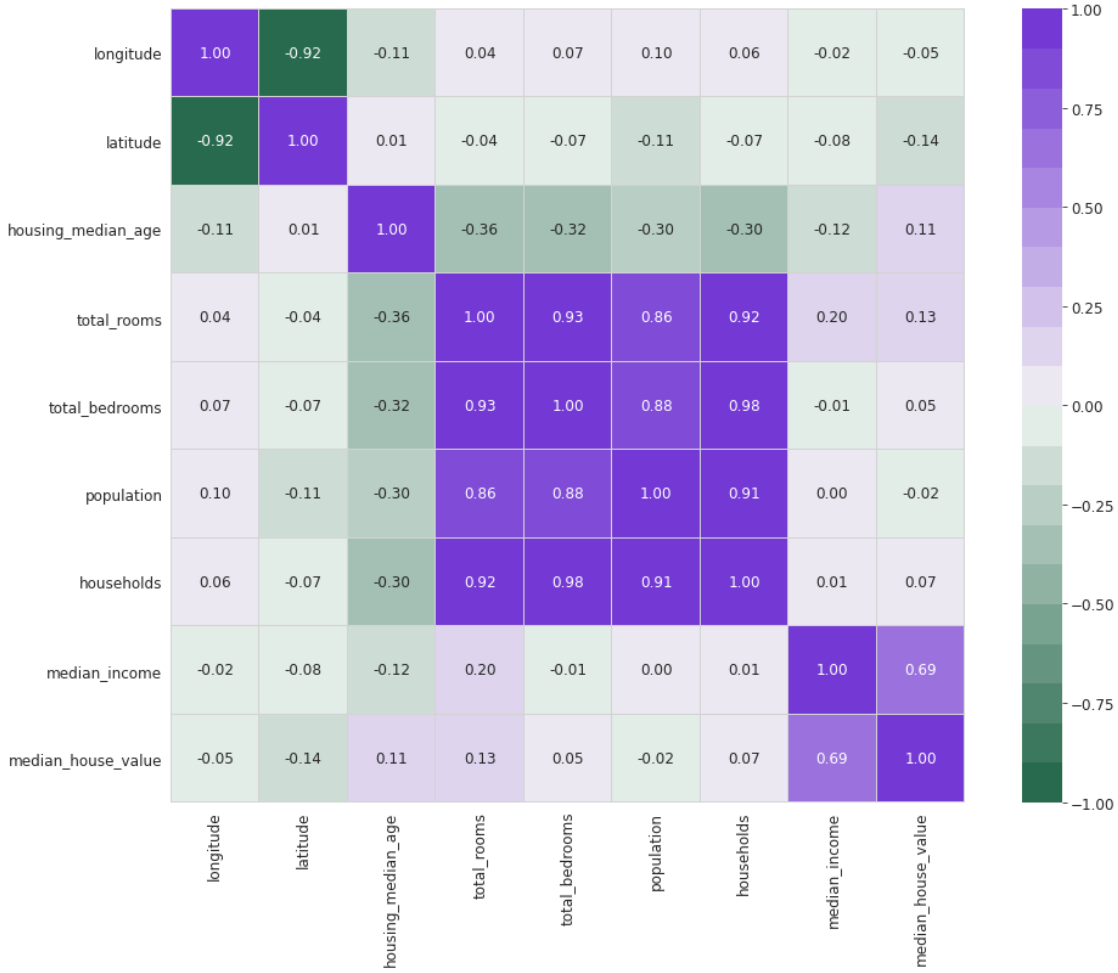

Figura 3

3 Metodología

3.1 Separación en sets de train y test

Empleamos la estrategia de muestreo estratificado utilizando la variable median_income, para no incluir un sesgo en el muestreo. Dividimos las observaciones en subgrupos homogéneos respetando la distribución que presenta el gráfico de la Figura 4. Este método garantiza que tanto el set de test y de train sean representativos de toda la población.

[10]:
```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
housing["income_cat"].hist()
plt.title("Distribucion de ingresos en 5 grupos")
```

[10]:
```
Text(0.5, 1.0, 'Distribucion de ingresos en 5 grupos')
```

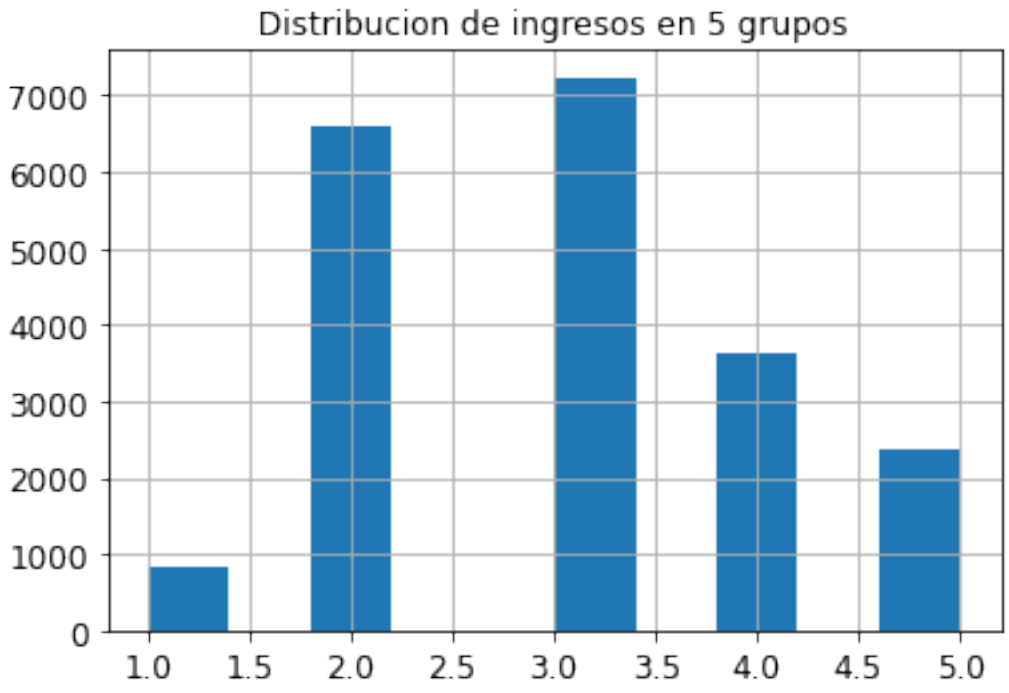

Figura 4

Comparamos las proporciones de la variable income_cat entre el data set completo, el set de testing resultado de un muestreo aleatorio y un muestreo estratificado. Vemos que el muestreo estratificado representa mejor las proporciones originales del set completo, por lo tanto elegimos este para realizar la división de datos.

[11]:
```
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]

def income_cat_proportions(data):
    return data["income_cat"].value_counts() / len(data)


train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

compare_props = pd.DataFrame({
    "Overall": income_cat_proportions(housing),
    "Stratified": income_cat_proportions(strat_test_set),
    "Random": income_cat_proportions(test_set),
}).sort_index()
compare_props["Rand. %error"] = 100 * compare_props["Random"] /␣
 ↪compare_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] /␣
 ↪compare_props["Overall"] - 100

compare_props
```

[11]:

	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

3.2 Procesamiento y limpieza de datos

[12]:
```
#sacamos la variable income_cat dado que sólo fue utilizada para dividir␣
 ↪estratificadamente el data set
strat_train_set = strat_train_set.drop("income_cat", axis=1)

strat_test_set = strat_test_set.drop("income_cat", axis=1)
```

Para la preparación de datos seguimos los siguientes pasos: * Limpieza de datos faltantes * Estandarización de covariables numéricas excepto 'longitude' y 'latitude' * Transformacion de variables 'total_rooms', 'total_bedrooms', 'population' y 'households' en rooms_per_household, population_per_household, bedrooms_per_room * Transformación de la variable categorica mediante la funcion *oneHotEncoder()* generando cinco variables numericas binarias (una por cada categoria).

Este pipeline de procesamiento fue aplicado tanto al set de train como al set de test.

[13]:
```
#esto lo hacemos para que el pipeline funcione correctamente. Simplemente␣
 ↪transforma el dataset en un ndArray
#para que luego pueda utilizarlo la transformacion con CombinedAttributesAdder
class ConvertToNDArray(BaseEstimator, TransformerMixin):
    def __init__(self): # no *args or **kargs
        return None
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        return X.values

convert_toNdArray = ConvertToNDArray()

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True,rooms_ix=2, bedrooms_ix=3,␣
 ↪population_ix=4, households_ix=5): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
        self.rooms_ix = rooms_ix
        self.bedrooms_ix=bedrooms_ix
        self.population_ix=population_ix
        self.households_ix=households_ix

    def fit(self, X, y=None):
        return self  # nothing else to do

    def transform(self, X):
        rooms_per_household = X[:, self.rooms_ix] / X[:, self.households_ix]
        population_per_household = X[:, self.population_ix] / X[:, self.
 ↪households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, self.bedrooms_ix] / X[:, self.rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            result = np.c_[X, rooms_per_household, population_per_household]
            return result
```

[14]:
```
def processingData(dataSet, targetFeature, categoricalFeature, col_names,␣
 ↪extra_attribs):
        #quitamos los nan
        dataSet_withoutnan = dataSet.dropna().copy()

        #dividimos el dataset en las variables que se usan en la prediccion y la␣
 ↪variable a predecir
        data = dataSet_withoutnan.drop(targetFeature, axis=1).copy() # drop␣
 ↪labels for training set
        data_labels = dataSet_withoutnan[targetFeature].copy()
```

```
        #procesamos primero los datos numericos
        print("procesando los datos numericos...\n")
        numerical_data = data.drop([categoricalFeature, "longitude",␣
 ↪"latitude"], axis=1).copy()

        #obtenemos los indices de las columnas que luego vamos a combinar
        rooms_ix, bedrooms_ix, population_ix, households_ix = [
            data.columns.get_loc(c) for c in col_names] # get the column indices

        #generamos el dataframe con los atributos extra
        attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=True)
        data_extra_attribs = attr_adder.transform(data.values)
        data_extra = pd.DataFrame(
            data_extra_attribs,
            columns=list(data.columns)+extra_attribs,
            index=data.index)

        num_pipeline = Pipeline([
        ('convert_toNdArray', ConvertToNDArray()),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
        ])

        #procesamos los datos categoricos
        print("procensando variables categoricas...\n")
        cat_attribs = [categoricalFeature]

        full_pipeline = ColumnTransformer([
                ("num", num_pipeline, list(numerical_data)),
                ("cat", OneHotEncoder(), cat_attribs),
            ])

        data_prepared = full_pipeline.fit_transform(data_extra.iloc[:, 2:])

        #devolvemos los datos procesados y los objetivos
        print("procesamiento finalizado :D")
        return data_prepared, data_labels, data_extra
```

[15]:
```
train, train_labels, data_processed = processingData(strat_train_set,␣
 ↪"median_house_value",
                "ocean_proximity",
                ["total_rooms", "total_bedrooms", "population", "households"],
                ["rooms_per_household",␣
 ↪"population_per_household","bedrooms_per_room"])
```

```
procesando los datos numericos...
```

```
procensando variables categoricas...

procesamiento finalizado :D
```

[16]:
```
test, test_labels, data_processed_test = processingData(strat_test_set,␣
↪"median_house_value",
                                        "ocean_proximity",
                                        ["total_rooms", "total_bedrooms",␣
↪"population", "households"],
                                        ["rooms_per_household",␣
↪"population_per_household","bedrooms_per_room"])
```

```
procesando los datos numericos...

procensando variables categoricas...

procesamiento finalizado :D
```

[17]:
```
def correlationPostTransform(data_prepared, df_extra_attr):
    combined_data=np.c_[df_extra_attr.iloc[:, 0:2], data_prepared].copy()

    names = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
           'total_bedrooms', 'population', 'households', 'median_income',
            'rooms_per_household','population_per_household',␣
↪'bedrooms_per_room']#11

    data_corr = pd.DataFrame(
        combined_data,
        columns=list(names)+["0", "1", "2", "3", "4"],#+5
        index=df_extra_attr.index).astype(float).copy()

    attributes = ['housing_median_age', 'total_rooms',
           'total_bedrooms', 'population', 'households', 'median_income',
            'rooms_per_household','population_per_household',␣
↪'bedrooms_per_room']

    axes = scatter_matrix(data_corr[attributes], alpha=0.2, hist_kwds={'bins':
↪30}, figsize=(12,8))
    for ax in axes.flatten():
        ax.xaxis.label.set_rotation(90)
        ax.yaxis.label.set_rotation(0)
        ax.yaxis.label.set_ha('right')
    plt.gcf().subplots_adjust(wspace=0, hspace=0)
    plt.show()

    corr_matrix_pr = data_corr.corr()
```

```
    paleta = sns.diverging_palette(150, 275, s=80, l=40, n=20)
    with sns.axes_style("white"):
        f, ax = plt.subplots(figsize=(17, 12))
        ax = sns.heatmap(corr_matrix_pr,
                         annot=True,
                         annot_kws={'size': 12},
                         fmt='.2f',
                         vmax=1,
                         vmin=-1,
                         square=True,
                         linewidths=.01,
                         linecolor='lightgray',
                         cmap=paleta)
    return combined_data, data_corr
```

La relación entre los atributos numéricos estandarizados incluyendo las variables transformadas, exceptuando 'longitude' y 'latitude' se observa en los *scatterplots* de las figuras 5 (train) y 7 (test). Los valores de correlación entre los atributos se puede observa en la figura 6 (train) y 8 (test).

[18]:
```
train_transf, data_train_transf = correlationPostTransform(train,data_processed)
```

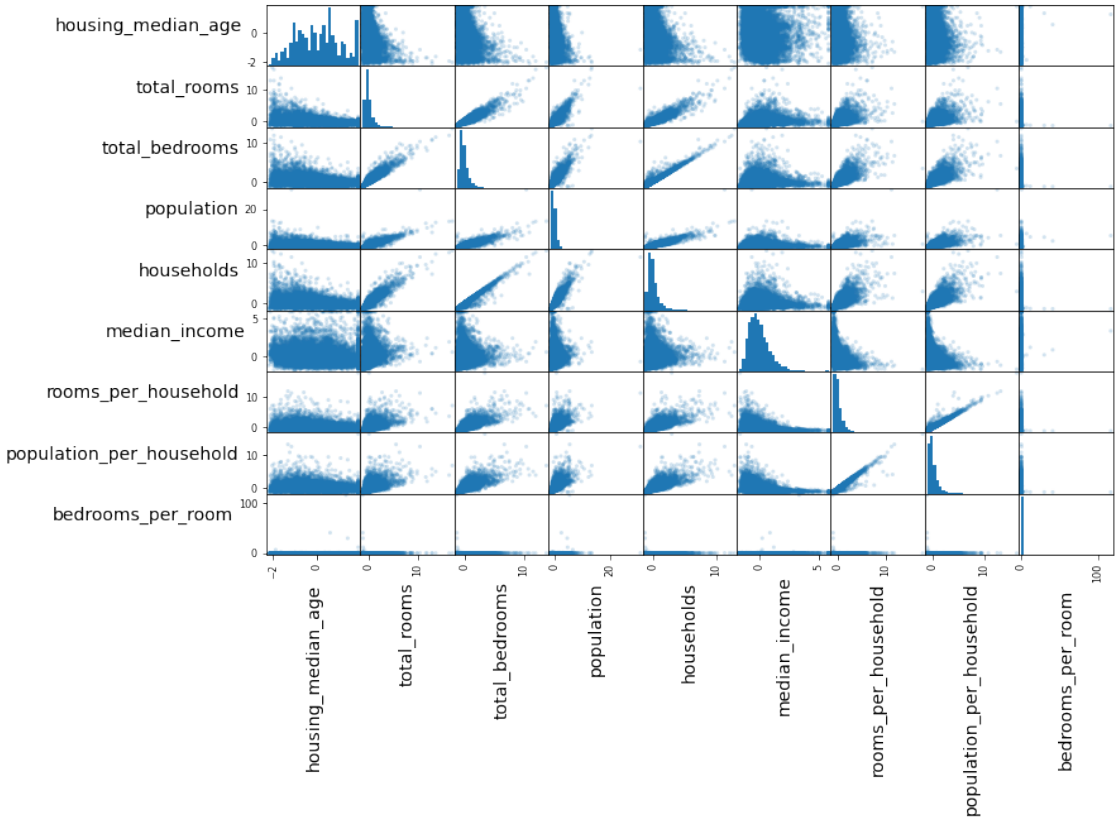

Figura 5

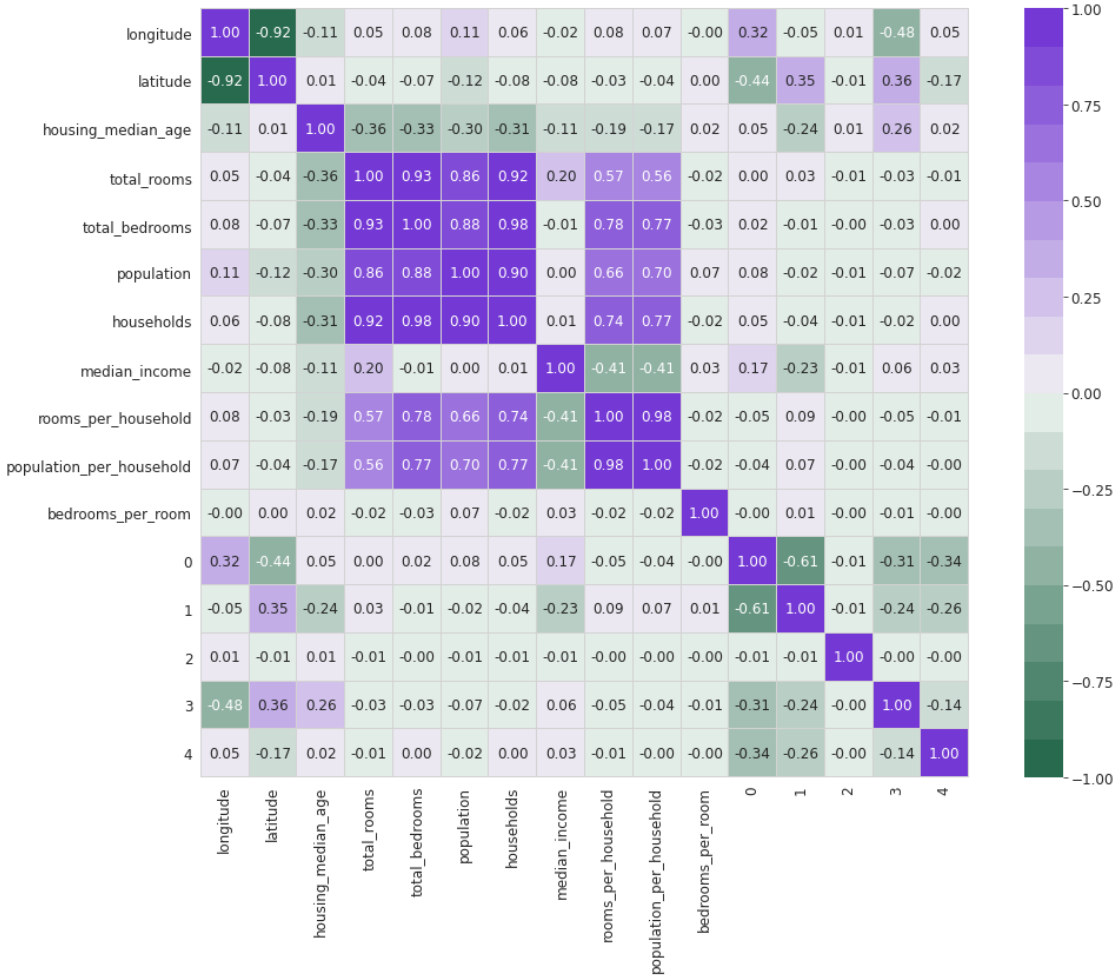

Figura 6

[19]:
```
test_transf, data_test_transf =␣
↪correlationPostTransform(test,data_processed_test)
```

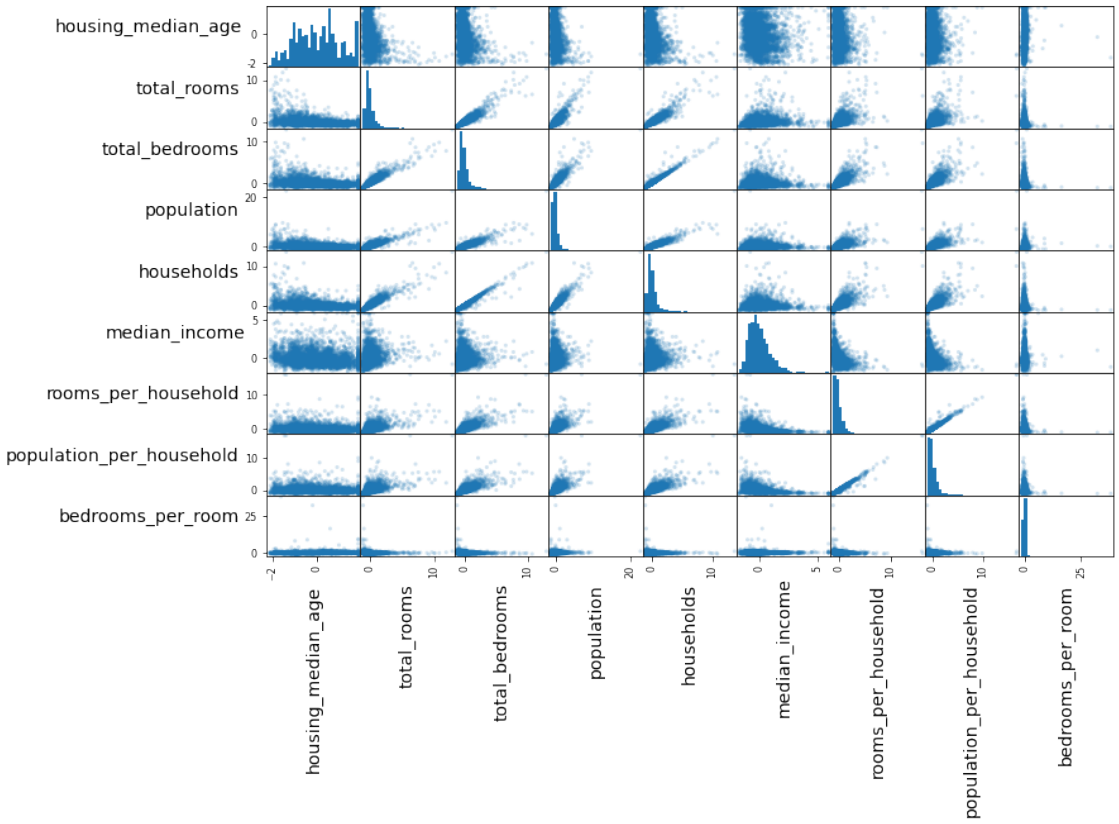
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
rooms_per_household
population_per_household
bedrooms_per_room

Figura 7

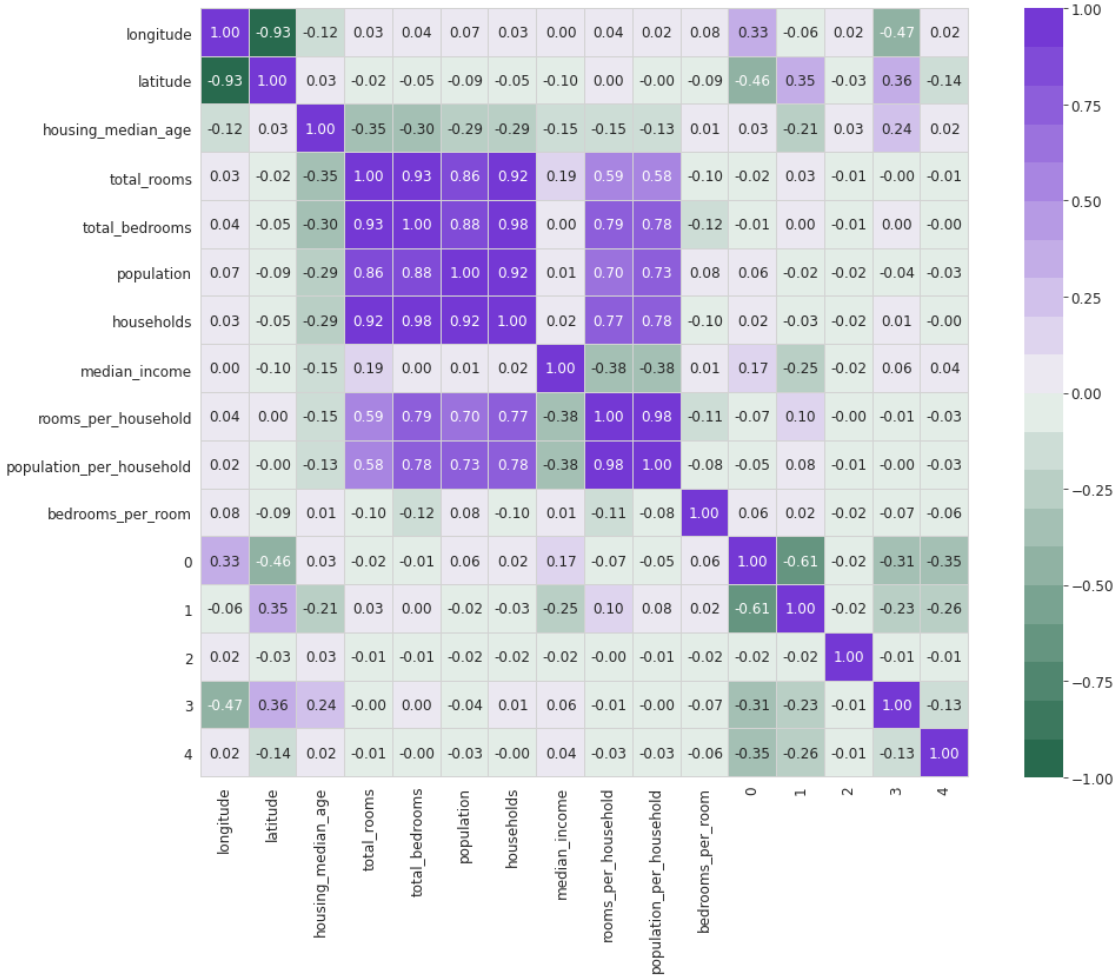

Figura 8

[40]:
```
def finalizacionDeProcesamiento_notTransf(data_prepared, df_extra_attr):
    combined_data=np.c_[df_extra_attr.iloc[:, 0:2], data_prepared].copy()

    names = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
           'total_bedrooms', 'population', 'households', 'median_income',
            'rooms_per_household','population_per_household',␣
↪'bedrooms_per_room']

    namesToDrop = ['rooms_per_household', 'bedrooms_per_room',␣
↪'population_per_household']


    data_corr = pd.DataFrame(
        combined_data,
        columns=list(names)+["0", "1", "2", "3", "4"],
```

```
        index=df_extra_attr.index).astype(float).drop(namesToDrop, axis=1).copy()

    corr_matrix_pr = data_corr.corr()
    paleta = sns.diverging_palette(150, 275, s=80, l=40, n=20)
    with sns.axes_style("white"):
        f, ax = plt.subplots(figsize=(17, 12))
        ax = sns.heatmap(corr_matrix_pr,
                         annot=True,
                         annot_kws={'size': 12},
                         fmt='.2f',
                         vmax=1,
                         vmin=-1,
                         square=True,
                         linewidths=.01,
                         linecolor='lightgray',
                         cmap=paleta)
    return combined_data, data_corr
```

Los valores de correlación entre los atributos numéricos estandarizados, exceptuando 'longitude' y 'latitude' se observa en las figuras 9 (train) y 10 (test).

[44]:
```
train_notTransf, data_train_notTransf =␣
 ↪finalizacionDeProcesamiento_notTransf(train,data_processed)
test_notTransf, data_test_notTransf =␣
 ↪finalizacionDeProcesamiento_notTransf(test,data_processed_test)
```

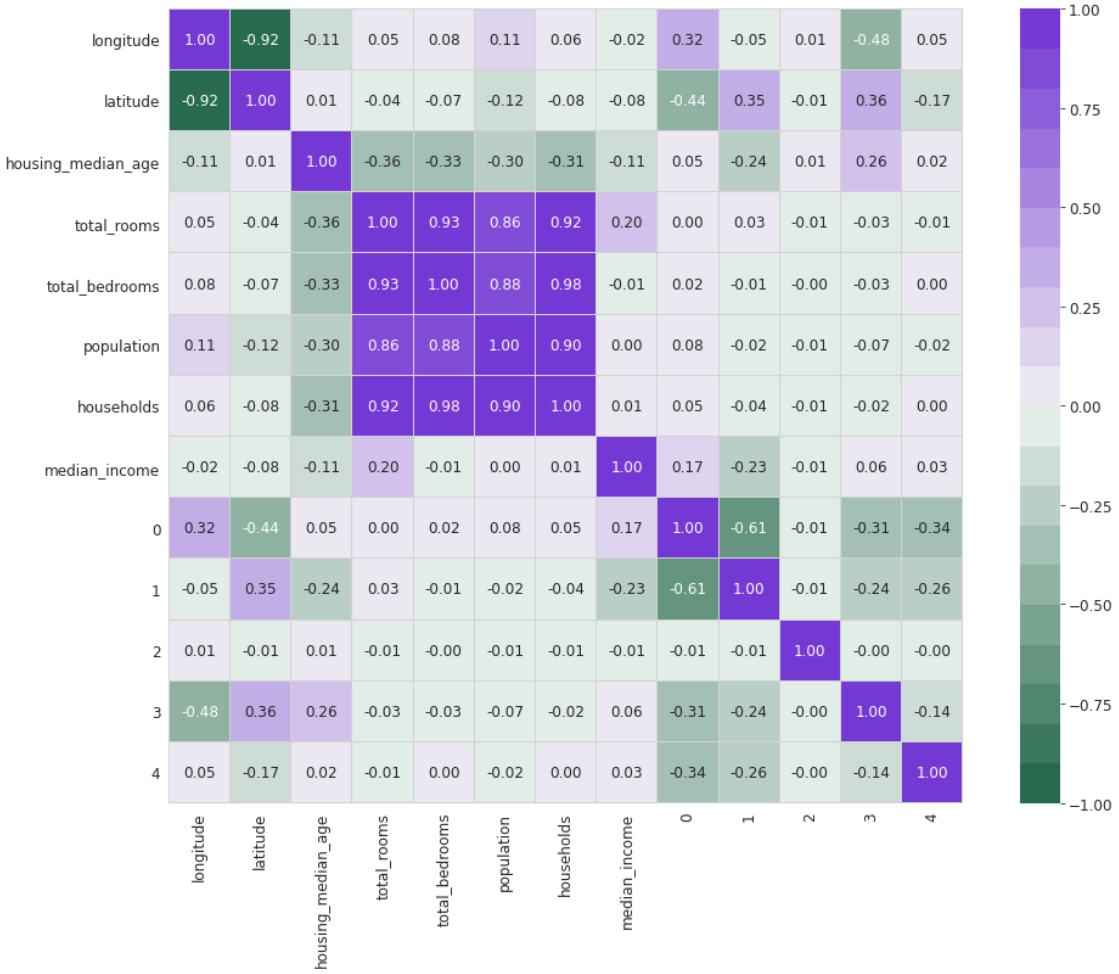
longitude 1.00 -0.92 -0.11 0.05 0.08 0.11 0.06 -0.02 0.32 -0.05 0.01 -0.48 0.05
latitude -0.92 1.00 0.01 -0.04 -0.07 -0.12 -0.08 -0.08 -0.44 0.35 -0.01 0.36 -0.17
housing_median_age -0.11 0.01 1.00 -0.36 -0.33 -0.30 -0.31 -0.11 0.05 -0.24 0.01 0.26 0.02
total_rooms 0.05 -0.04 -0.36 1.00 0.93 0.86 0.92 0.20 0.00 0.03 -0.01 -0.03 -0.01
total_bedrooms 0.08 -0.07 -0.33 0.93 1.00 0.88 0.98 -0.01 0.02 -0.01 -0.00 -0.03 0.00
population 0.11 -0.12 -0.30 0.86 0.88 1.00 0.90 0.00 0.08 -0.02 -0.01 -0.07 -0.02
households 0.06 -0.08 -0.31 0.92 0.98 0.90 1.00 0.01 0.05 -0.04 -0.01 -0.02 0.00
median_income -0.02 -0.08 -0.11 0.20 -0.01 0.00 0.01 1.00 0.17 -0.23 -0.01 0.06 0.03
0 0.32 -0.44 0.05 0.00 0.02 0.08 0.05 0.17 1.00 -0.61 -0.01 -0.31 -0.34
1 -0.05 0.35 -0.24 0.03 -0.01 -0.02 -0.04 -0.23 -0.61 1.00 -0.01 -0.24 -0.26
2 0.01 -0.01 0.01 -0.01 -0.00 -0.01 -0.01 -0.01 -0.01 -0.01 1.00 -0.00 -0.00
3 -0.48 0.36 0.26 -0.03 -0.03 -0.07 -0.02 0.06 -0.31 -0.24 -0.00 1.00 -0.14
4 0.05 -0.17 0.02 -0.01 0.00 -0.02 0.00 0.03 -0.34 -0.26 -0.00 -0.14 1.00
longitude
latitude
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
0
1
2
3
4
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Figura 9

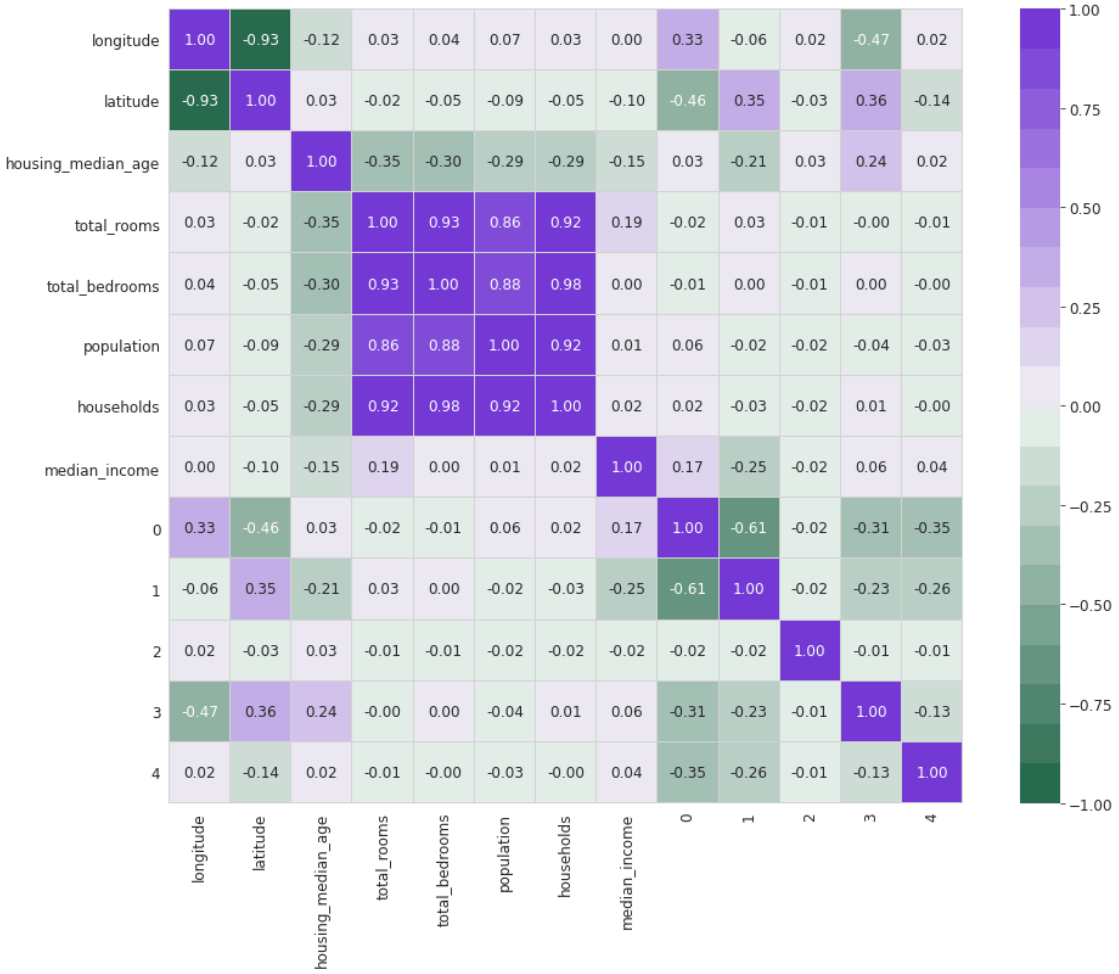

Figura 10

[46]:
```
def finalizacionDeProcesamiento_drop(data_prepared, df_extra_attr):
    combined_data=np.c_[df_extra_attr.iloc[:, 0:2], data_prepared].copy()

    names = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
           'total_bedrooms', 'population', 'households', 'median_income',
            'rooms_per_household','population_per_household',␣
 ↪'bedrooms_per_room']
    namesToDrop = ['population', 'households', 'total_rooms', 'total_bedrooms',
                   'bedrooms_per_room', 'rooms_per_household']

    data_corr = pd.DataFrame(
        combined_data,
        columns=list(names)+["0", "1", "2", "3", "4"],
        index=df_extra_attr.index).astype(float).drop(namesToDrop, axis=1).copy()
```

```
    corr_matrix_pr = data_corr.corr()
    paleta = sns.diverging_palette(150, 275, s=80, l=40, n=20)
    with sns.axes_style("white"):
        f, ax = plt.subplots(figsize=(17, 12))
        ax = sns.heatmap(corr_matrix_pr,
                         annot=True,
                         annot_kws={'size': 12},
                         fmt='.2f',
                         vmax=1,
                         vmin=-1,
                         square=True,
                         linewidths=.01,
                         linecolor='lightgray',
                         cmap=paleta)
    return combined_data, data_corr
```

Los valores de correlación entre los atributos numéricos estandarizados, exceptuando 'longitude' y 'latitude' se observa en las figuras 11 (train) y 12 (test). En este caso se eliminaron los atributos que se utilizaron para las transformaciones y dos de las features transformadas.

[47]:
```
train_dropped, data_train_dropped =␣
 ↪finalizacionDeProcesamiento_drop(train,data_processed)
test_dropped, data_test_dropped =␣
 ↪finalizacionDeProcesamiento_drop(test,data_processed_test)
```

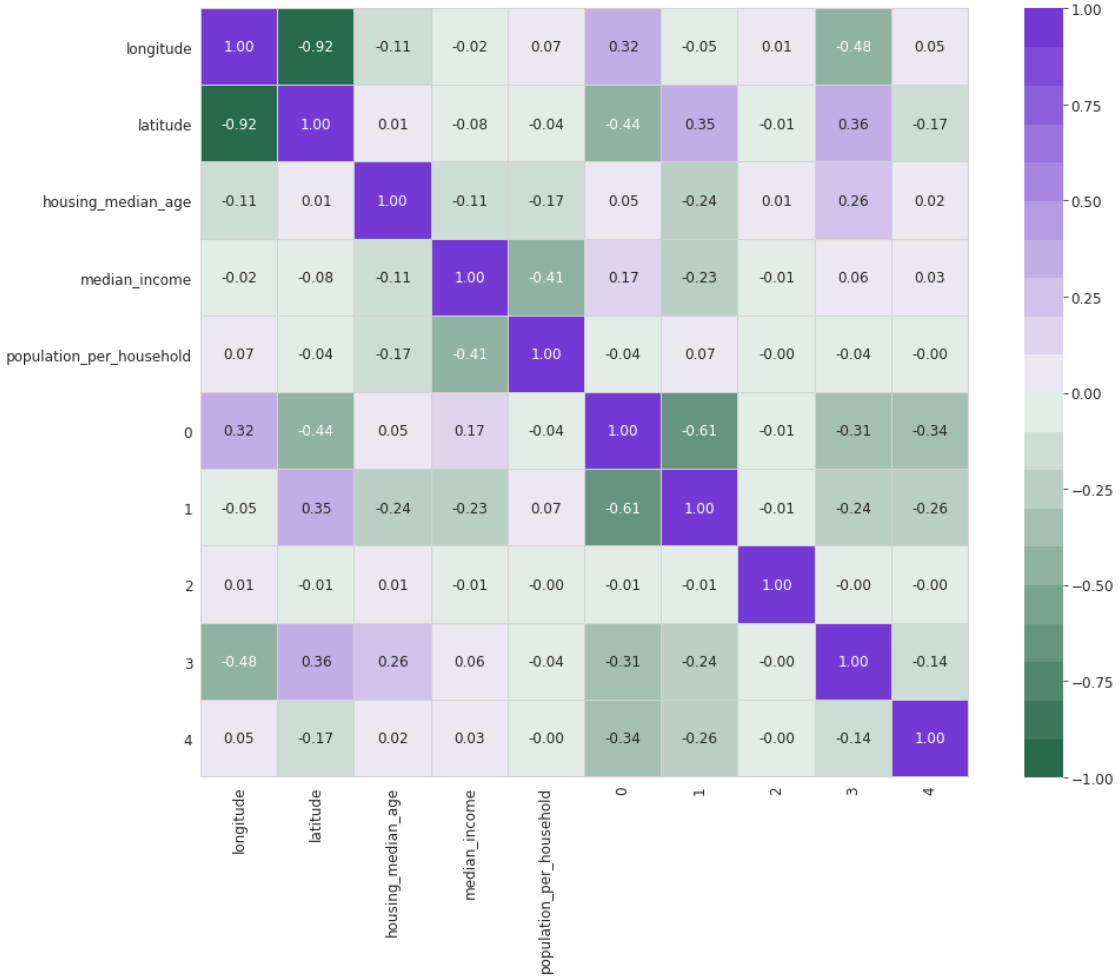
longitude 1.00 -0.92 -0.11 -0.02 0.07 0.32 -0.05 0.01 -0.48 0.05
latitude -0.92 1.00 0.01 -0.08 -0.04 -0.44 0.35 -0.01 0.36 -0.17
housing_median_age -0.11 0.01 1.00 -0.11 -0.17 0.05 -0.24 0.01 0.26 0.02
median_income -0.02 -0.08 -0.11 1.00 -0.41 0.17 -0.23 -0.01 0.06 0.03
population_per_household 0.07 -0.04 -0.17 -0.41 1.00 -0.04 0.07 -0.00 -0.04 -0.00
0 0.32 -0.44 0.05 0.17 -0.04 1.00 -0.61 -0.01 -0.31 -0.34
1 -0.05 0.35 -0.24 -0.23 0.07 -0.61 1.00 -0.01 -0.24 -0.26
2 0.01 -0.01 0.01 -0.01 -0.00 -0.01 -0.01 1.00 -0.00 -0.00
3 -0.48 0.36 0.26 0.06 -0.04 -0.31 -0.24 -0.00 1.00 -0.14
4 0.05 -0.17 0.02 0.03 -0.00 -0.34 -0.26 -0.00 -0.14 1.00
longitude
latitude
housing_median_age
median_income
population_per_household
0
1
2
3
4
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Figura 11

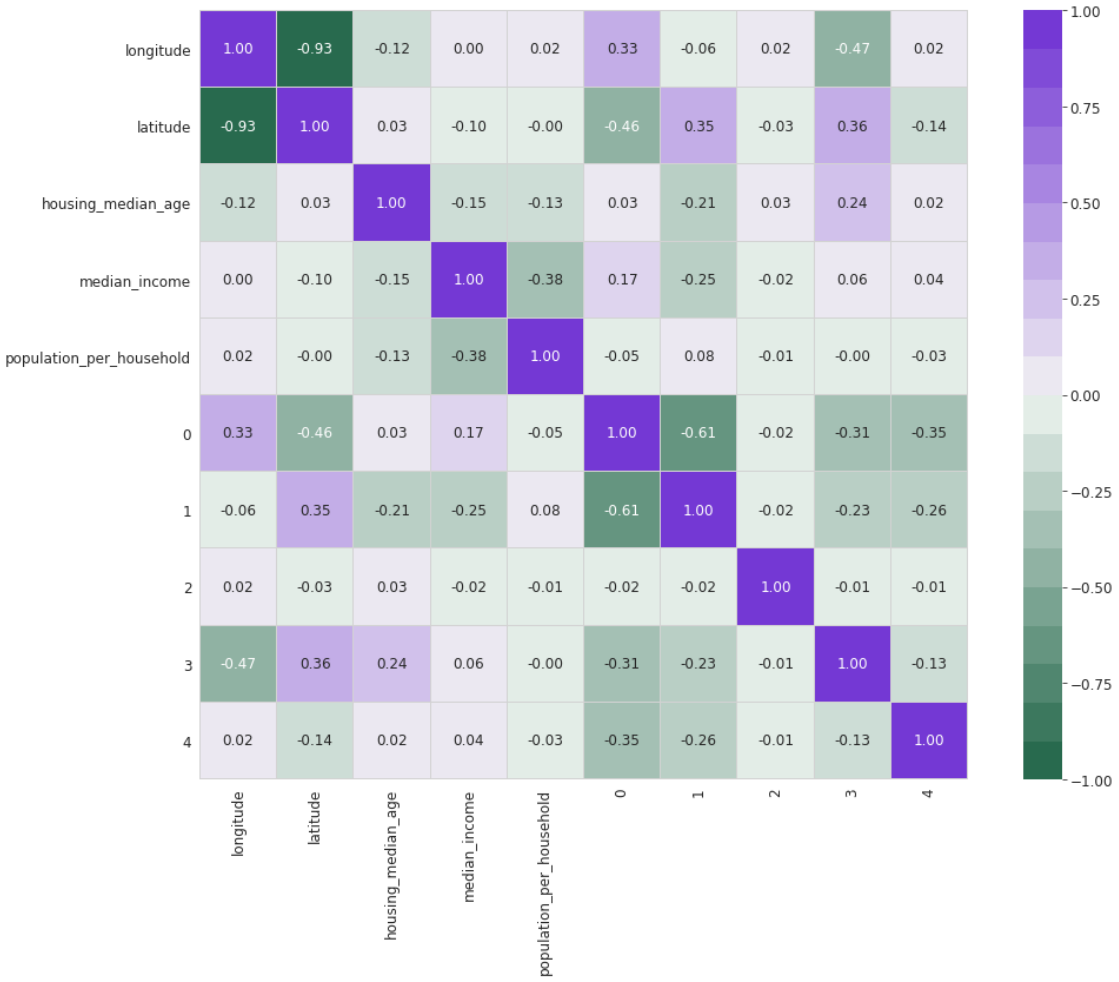

Figura 12

3.3 Entrenamiento de los modelos

[21]:
```
def selectValuesForHyperparameter(initialValue, lastValue, jumps):
    rv = []
    for v in range(int(initialValue*100), int(lastValue*100),int(jumps*100)):
        rv.append(float(v/100));
    return rv

def models(train, train_labels, test, test_labels):

    alphaValues = selectValuesForHyperparameter(0.8, 3, 0.2)

    #LASSO

    #búsqueda de hiperpatámetro
```

```
    lasso_params = {'alpha':alphaValues}
    lasso = linear_model.Lasso(max_iter=5000)
    clf_lasso = GridSearchCV(lasso, lasso_params, cv=10,
                                    scoring='neg_mean_squared_error',
                                    return_train_score=True)
    clf_lasso.fit(train, train_labels)
    print('El valor del hiperparámetro seleccionado para LASSO es:
↪'+str(clf_lasso.best_params_['alpha'])+'\n')

    #train
    lasso = linear_model.LassoCV(alphas=[clf_lasso.best_params_['alpha']],
                                 max_iter=5000,
                                 cv=10,
                                 random_state=0)
    lasso.fit(train, train_labels)

    #test
    housing_predictions_lasso = lasso.predict(test)
    lin_mse_lasso = mean_squared_error(test_labels, housing_predictions_lasso)
    lin_rmse_lasso = np.sqrt(lin_mse_lasso)



    #RIDGE

    #búsqueda de hiperpatámetro
    alphaValues = selectValuesForHyperparameter(1, 10, 0.1)
    ridge_params = {'alpha':alphaValues}
    ridge = Ridge(max_iter=5000)
    clf_ridge = GridSearchCV(ridge, ridge_params, cv=10,
                                        scoring='neg_mean_squared_error',
                                        return_train_score=True)
    clf_ridge.fit(train, train_labels)
    print('El valor del hiperparámetro seleccionado para Ridge es:
↪'+str(clf_ridge.best_params_['alpha'])+'\n')

    #train
    ridge = linear_model.RidgeCV(alphas=[clf_ridge.best_params_['alpha']],
                                 cv=10,
                                 scoring='neg_mean_squared_error')
    ridge.fit(train, train_labels)

    #test
    housing_predictions_ridge = ridge.predict(test)
    lin_mse_ridge = mean_squared_error(test_labels, housing_predictions_ridge)
    lin_rmse_ridge = np.sqrt(lin_mse_ridge)
```

```
    #ELASTIC NET

    #búsqueda de hiperpatámetro
    alphaValues = selectValuesForHyperparameter(0.2, 10, 0.2);
    l1Values = selectValuesForHyperparameter(0.05, 1, 0.05);

    elastic_params = {'alpha':alphaValues, 'l1_ratio':l1Values}
    elastic = linear_model.ElasticNet(max_iter=5000)
    clf_elastic = GridSearchCV(elastic, elastic_params, cv=10,
                                        scoring='neg_mean_squared_error',
                                        return_train_score=True)
    clf_elastic.fit(train, train_labels)
    print('El valor de los hiperparámetros seleccionados para Elastic Net es:␣
↪alpha:'+str(clf_elastic.best_params_['alpha'])+', l1:'+str(clf_elastic.
↪best_params_['l1_ratio'])+'\n')

    #train
    elastic = linear_model.ElasticNetCV(alphas=[clf_elastic.
↪best_params_['alpha']],
                                    l1_ratio=[clf_elastic.
↪best_params_['l1_ratio']],
                                    max_iter=5000,
                                    cv=10)
    elastic.fit(train, train_labels)

    #test
    housing_predictions_elastic = elastic.predict(test)
    lin_mse_elastic = mean_squared_error(test_labels,␣
↪housing_predictions_elastic)
    lin_rmse_elastic = np.sqrt(lin_mse_elastic)

    #Comparación de los RMSE para los tres modelos
    print("El RMSE para Lasso es:"+str(lin_rmse_lasso)+'\n')
    print("El RMSE para Ridge es:"+str(lin_rmse_ridge)+'\n')
    print("El RMSE para Elastic Net es:"+str(lin_rmse_elastic)+'\n')
```

4 Resultados y Discusión

[45]:
```
# 1
models(data_train_notTransf, train_labels, data_test_notTransf, test_labels)
```

```
El valor del hiperparámetro seleccionado para LASSO es:0.8

El valor del hiperparámetro seleccionado para Ridge es:1.0
```

```
El valor de los hiperparámetros seleccionados para Elastic Net es: alpha:0.2,
l1:0.95

El RMSE para Lasso es:67236.40792328252

El RMSE para Ridge es:67259.11714088148

El RMSE para Elastic Net es:67458.41749028569
```

[22]:
```
# 2
models(train_transf, train_labels, test_transf, test_labels)
```

```
El valor del hiperparámetro seleccionado para LASSO es:0.8

El valor del hiperparámetro seleccionado para Ridge es:1.0

El valor de los hiperparámetros seleccionados para Elastic Net es: alpha:0.2,
l1:0.95

El RMSE para Lasso es:67288.37743804531

El RMSE para Ridge es:67310.25810605213

El RMSE para Elastic Net es:67457.45464337643
```

[48]:
```
# 3
models(data_train_dropped, train_labels, data_test_dropped, test_labels)
```

```
El valor del hiperparámetro seleccionado para LASSO es:0.8

El valor del hiperparámetro seleccionado para Ridge es:1.0

El valor de los hiperparámetros seleccionados para Elastic Net es: alpha:0.2,
l1:0.95

El RMSE para Lasso es:70445.1158831168

El RMSE para Ridge es:70472.70611634907

El RMSE para Elastic Net es:70642.7491364116
```

Se compararon los tres modelos lineales con tres selecciones de variables distintas:

1) Features originales (Figura 9 y 10)

2) Features originales + nuevas features calculadas a partir de 4 de las originales. (Figuras 6 y 8)

3) Una selección acotada de features originales y seleccionando sólo una de los nuevos atributos. (Figuras 11 y 12)

Observamos que los valores más pequeños de RMSE se obtuvieron para el data-set con los atributos originales (1). Comparando los valores de los RMSE obtenidos para los tres modelos, el mejor resultó ser LASSO para las tres selecciones de atributos que se utilizaron.