Evidencia Machine Learning

Camila Cusicanqui A00571258

18 de Septiembre de 2022

En este reporte se realizará un análisis sobre el desempeño de un set de datos en el cual generamos un indicadores claros y gráficas comparativas para respaldar nuestro análisis. Este set de datos proviene de Kaggle: https://www.kaggle.com/datasets/quantbruce/real-estate-price-prediction . Estos datos repressentan una relación entre los variable independientes, X_i y Y, nuestra variable dependiente como el precio del mercado de casas. Presenta las siguientes características:

- Total no. de attributes: 7
- Tamaño de muestra (n): 414
- Variables:
  - X_1: Fecha de transacción
  - X_2: Edad de casa
  - X_3: distancia a la estación de metro más cercana
  - X_4: Número de tiendas cercanas
  - X_5: Latitud
  - X_6: Longitud
  - Y: precio por unidad de área

El análisis tendrá los siguientes elementos:

- Separación y evaluación del modelo con un conjunto de prueba y un conjunto de validación (Train/Test/Validation).
- Diagnóstico y explicación el grado de bias o sesgo: bajo medio alto
- Diagnóstico y explicación el grado de varianza: bajo medio alto
- Diagnóstico y explicación el nivel de ajuste del modelo: underfitt fitt overfitt

Considerando el análisis previo utilizaremos técnicas de regularización o ajuste de parámetros para mejor nuestro modelo.

Exploración de datos

En está sección realizaremos una exploración de nuestros datos. Es decir, fijarnos en que no hay valores NaNs, nulos o repetidos.

In [1]:
```
import pandas as pd                # For data loading
import matplotlib.pyplot as plt    # For plots
import numpy as np                 # For arrays and math
import seaborn as sns              # For easier better looking plots
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import f1_score, r2_score, mean_squared_error, mean_abs
```

In [2]:
```
col = ['fecha','edad casa','metro','tiendas','lat','long','price']
df = pd.read_csv('Real Estate.csv',index_col = 'No')
df.columns = col
```

In [3]:
```
df.head()
```

Out[3]:

No	fecha	edad casa	metro	tiendas	lat	long	price
1	2012.917	32.0	84.87882	10	24.98298	121.54024	37.9
2	2012.917	19.5	306.59470	9	24.98034	121.53951	42.2
3	2013.583	13.3	561.98450	5	24.98746	121.54391	47.3
4	2013.500	13.3	561.98450	5	24.98746	121.54391	54.8
5	2012.833	5.0	390.56840	5	24.97937	121.54245	43.1

In [4]:
```
df.shape
```

Out[4]:
```
(414, 7)
```

In [5]:
```
df.isna().sum()
```

Out[5]:
```
fecha        0
edad casa    0
metro        0
tiendas      0
lat          0
long         0
price        0
dtype: int64
```

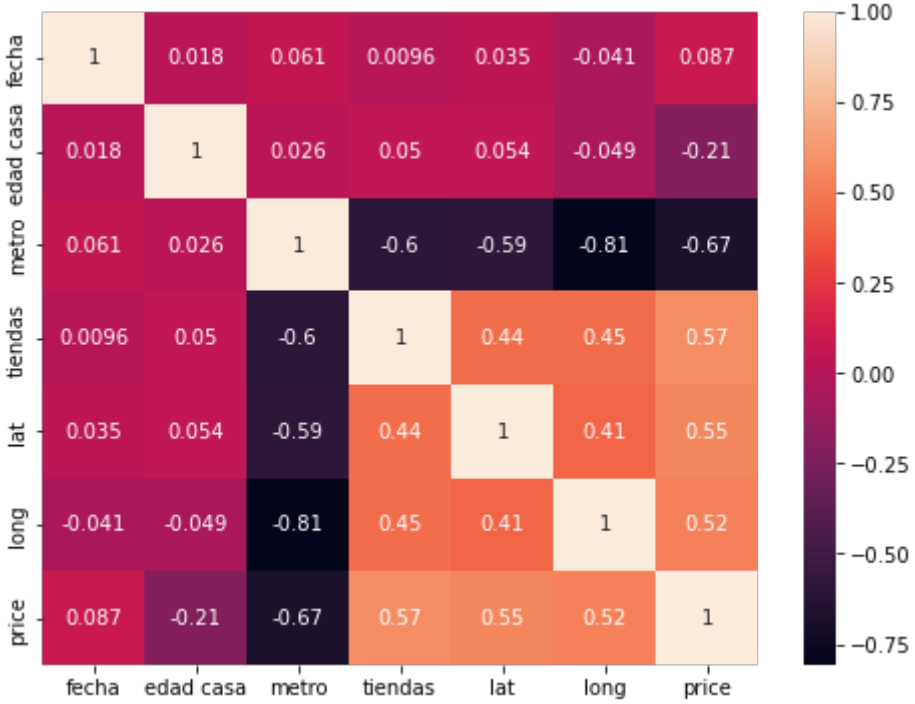
In [6]:
```
df = df.dropna()
```

In [7]:
```
# Queremos observar que tanta correlación hay entre los datos
print(pd.DataFrame(df).corr)
```

```
<bound method DataFrame.corr of         fecha  edad casa       metro  tiend
as       lat       long  price
No
1    2012.917       32.0    84.87882       10  24.98298  121.54024   37.9
2    2012.917       19.5   306.59470        9  24.98034  121.53951   42.2
3    2013.583       13.3   561.98450        5  24.98746  121.54391   47.3
4    2013.500       13.3   561.98450        5  24.98746  121.54391   54.8
5    2012.833        5.0   390.56840        5  24.97937  121.54245   43.1
..        ...        ...         ...      ...       ...        ...    ...
410  2013.000       13.7  4082.01500        0  24.94155  121.50381   15.4
411  2012.667        5.6    90.45606        9  24.97433  121.54310   50.0
412  2013.250       18.8   390.96960        7  24.97923  121.53986   40.6
413  2013.000        8.1   104.81010        5  24.96674  121.54067   52.5
414  2013.500        6.5    90.45606        9  24.97433  121.54310   63.9

[414 rows x 7 columns]>
```

In [8]:
```
plt.figure( figsize = (20,14) )
corr_df=df.corr(method='pearson')

plt.figure(figsize=(8, 6))
sns.heatmap(corr_df, annot=True)
plt.show()
plt.savefig('matriz_corr.png')
```

```
<Figure size 1440x1008 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

Observamos que para el precio la fecha casi no está relacionado con la fecha de transacción por está razón lo eliminaremos. Reducimos las dimensiones.

In [9]:
```
df = df.drop('fecha',axis = 1)
```

In [10]:
```
df.head()
```

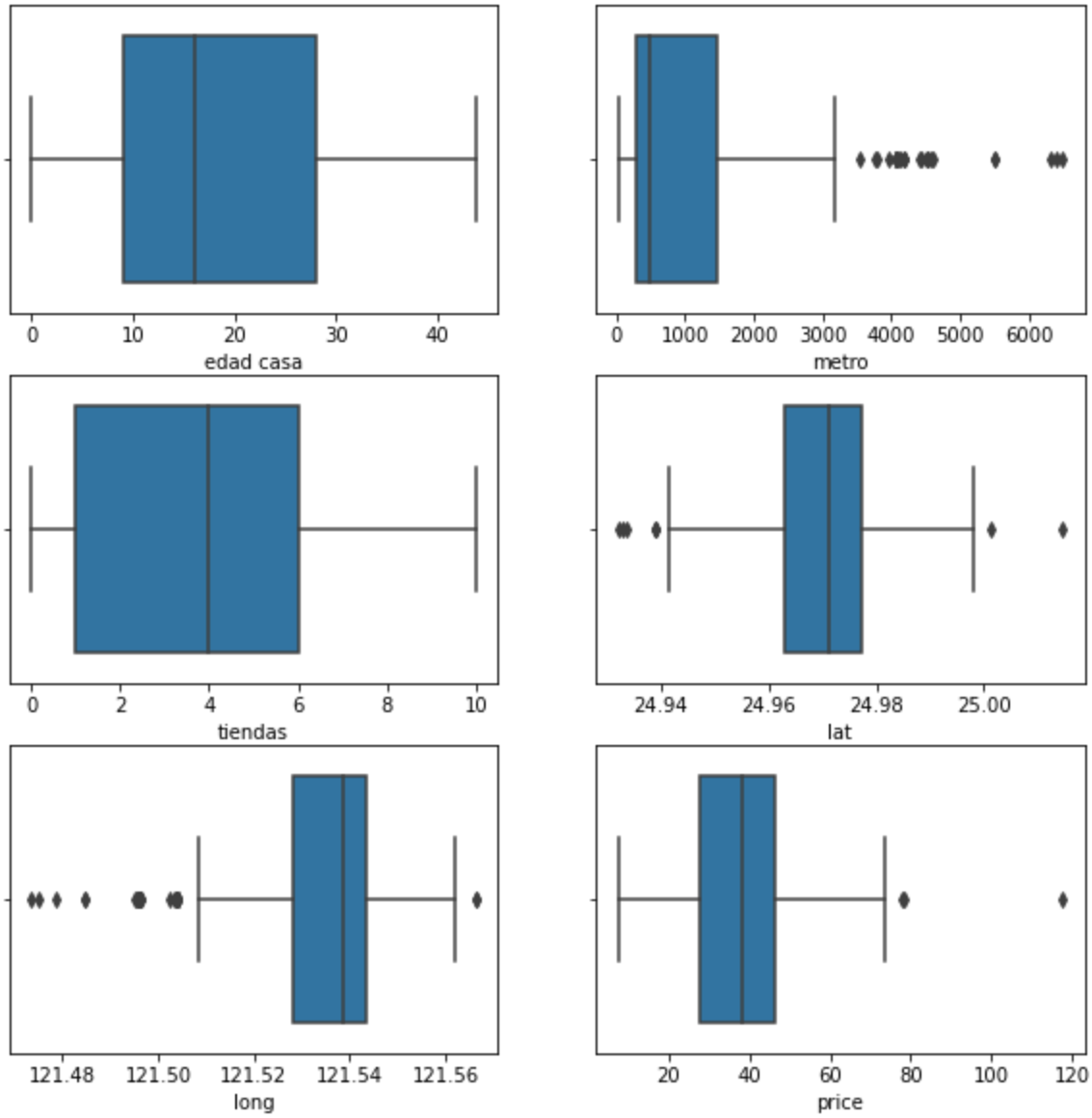
Out[10]:

No	edad casa	metro	tiendas	lat	long	price
1	32.0	84.87882	10	24.98298	121.54024	37.9
2	19.5	306.59470	9	24.98034	121.53951	42.2
3	13.3	561.98450	5	24.98746	121.54391	47.3
4	13.3	561.98450	5	24.98746	121.54391	54.8
5	5.0	390.56840	5	24.97937	121.54245	43.1

Realizaremos un análisis de los datos para observar si existen datos afuera del rango establecido.

In [11]:
```
# Figuras con boxplot para ver si hay datos atípicos
fig, axes = plt.subplots(3,2, figsize = (10,10) )
sns.boxplot(x= 'edad casa', data = df, ax = axes[0,0])
sns.boxplot(x= 'metro',  data = df, ax = axes[0,1])
sns.boxplot(x= 'tiendas', data = df, ax = axes[1,0])
sns.boxplot(x= 'lat', data = df, ax = axes[1,1])
sns.boxplot(x= 'long', data = df, ax = axes[2,0])
sns.boxplot(x= 'price', data = df, ax = axes[2,1])
```

Out[11]:
```
<AxesSubplot:xlabel='price'>
```

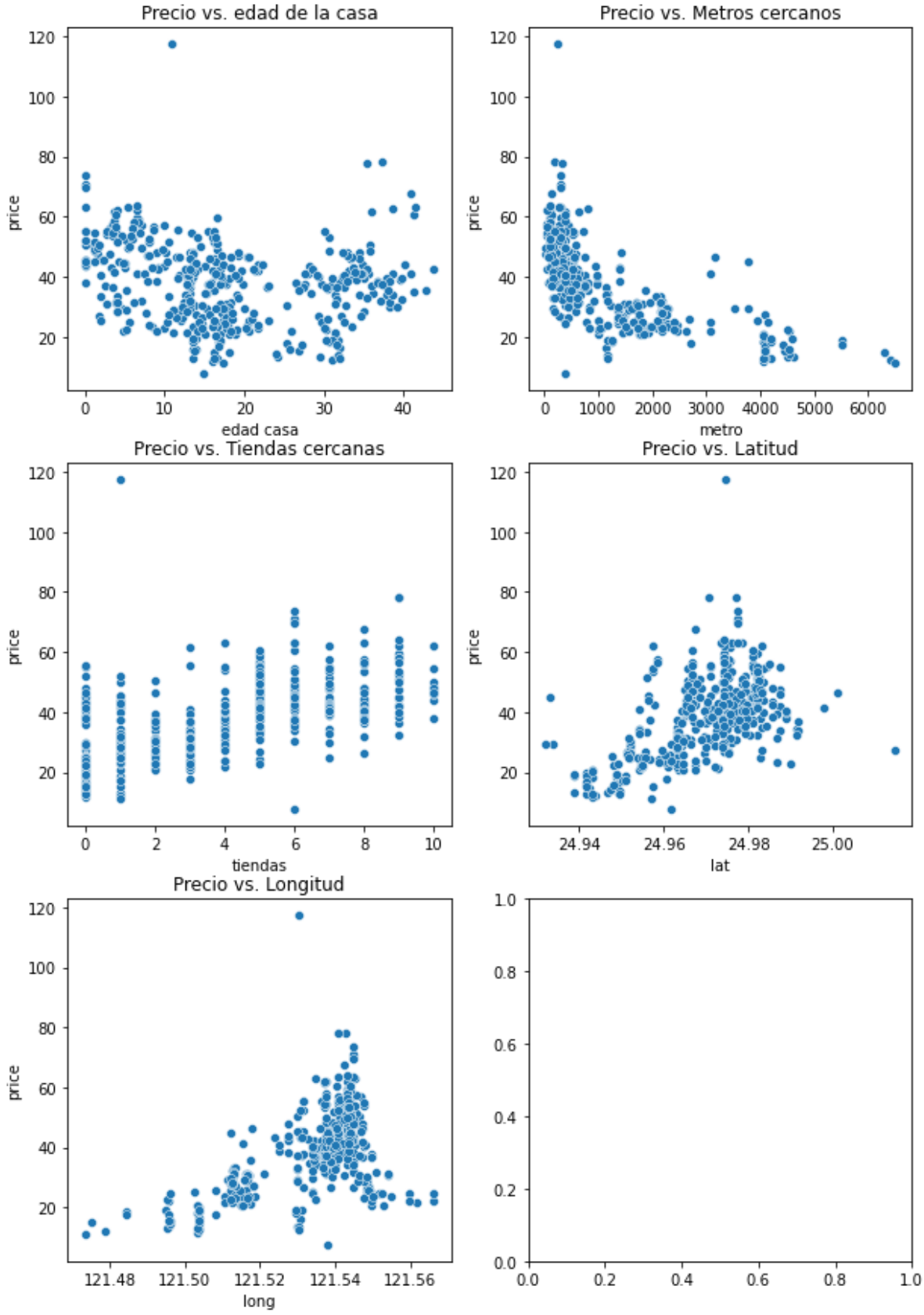


Visualización de los datos utilizando scatterplot

Deseamos visualizar los datos para asegurarnos que el set de datos se asimila a un comportamiento lineal.

In [12]:
```
fig, axes = plt.subplots(3,2, figsize = (10,15) )
sns.scatterplot(x= 'edad casa',y='price', data = df, ax = axes[0,0]).set(tit
sns.scatterplot(x= 'metro',y='price',  data = df, ax = axes[0,1]).set(title=
sns.scatterplot(x= 'tiendas',y='price', data = df, ax = axes[1,0]).set(title
sns.scatterplot(x= 'lat',y='price', data = df, ax = axes[1,1]).set(title='Pr
sns.scatterplot(x= 'long',y='price', data = df, ax = axes[2,0]).set(title='P
```

Out[12]:
```
[Text(0.5, 1.0, 'Precio vs. Longitud')]
```



Efectivamente, el comportamiento de los datos es apto para un modelo de regresión lineal. Podemos continuar con la implementación del modelo de regresión lineal.

Implementación de Máquina de aprendizaje

Ahora que ya tenemos nuestros datos selecionados podemos aplicar un modelo de regresión lineal multiple.

** Explicar por qué utilizamos regresión lineal multiple

Separación de datos de entrenamiento y prueba

Para construir el modelo se realiza una separación del conjunto de datos en 2 sets diferentes. El primer set se constituye en los datos de entrenamiento para el modelo. El segundo set tiene como propósito evaluar la efectividad de nuestro modelo con datos que previamente no había visto. Con esto podemos observar el desempeño de nuestro modelo de regresión lineal.

In [13]:
```
# Separación de la base de datos en un set de entrenamiento y evaluación
df_x = df.drop('price', axis = 1)
df_y = df['price']
testp = 0.25
seed1 = 1800

x1_train,x1_test,y1_train,y1_test = train_test_split(df_x, df_y, test_size =
```

75% de nuestro set de datos se utilizarán para entrenar y el resto (25%) se utilizará para comprobar la eficiencia de nuestro modelo de regresión lineal. Es importante recalcar que el `random_state` se utilizó con el objetivo de realizar todo este proyecto replicable. No hubo alguna modificación en los hiper parámetros al momento de integra `Regresión Lineal` .

In [14]:
```
#Establecemos nuestro modelo de regresión lineal
model = LinearRegression()
# Ajustamos nuestro modelo a los datos de entrenamiento
model.fit(x1_train,y1_train)
```

Out[14]:
```
LinearRegression()
```

Al entrenar nuestro modelo de regresión lineal ahora comienzamos a predecir valores de Y.

Para observar la varianza vemos la diferencia entre nuestra y pred y y entrenamiento.

In [15]:
```
y_pred_train = model.predict(x1_train)
```

Realizamos los valores de precisión pertinentes.

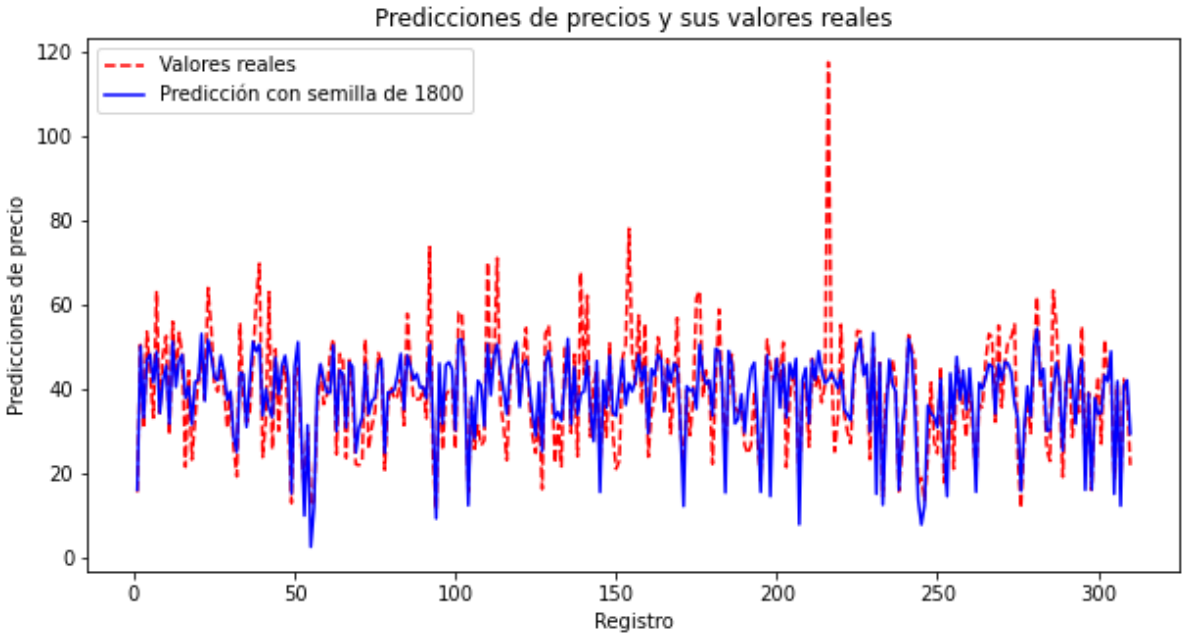
In [16]:
```
mse = mean_squared_error(y_pred_train, y1_train)
mae = mean_absolute_error(y_pred_train, y1_train)
r2 = r2_score(y_pred_train, y1_train)

print('Resultados de las predicciones con el set de entrenamiento:')
print('MSE con el conjunto de entrenamiento:', mse)
print('MAE con el conjunto de entrenamiento:', mae)
print('El coeficiente R^2:', r2)
```

```
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento: 83.32916191720045
MAE con el conjunto de entrenamiento: 6.264045629506458
El coeficiente R^2: 0.16684062426724977
```

Al entrenar nuestro regresor y después realizar predicciones con el mismo set de entrenamiento observamos errores considerables. Adicionalmente, nuestro coeficiente de determinación r^2 es muy bajo por lo que indica que nuestro modelo no es adecuado ya que un modelo con buen ajuste se aproxima a 1.

El sesgo y varianza lo podemos evaluar con la gráfica siguiente el cual muestra las predicciones realizadas por nuestro modelo en comparación a los valores reales del conjunto de entrenamiento. El sesgo es muy alta ya que las curvas no se parece, sin embargo, para evaluar la variación se debe implementar predicciones con distintas semillas para observar la variación real entre los modelos lo cual veremos próximamente.

In [17]:
```
xlab = np.arange(1,311)
plt.figure(figsize = (10,5))
plt.plot(xlab, y1_train, '--r',  label = 'Valores reales')
plt.plot(xlab, y_pred_train, '-b', label = 'Predicción con semilla de 1800')
plt.legend()
plt.title('Predicciones de precios y sus valores reales')
plt.xlabel('Registro')
plt.ylabel('Predicciones de precio')
plt.show()
```



Ahora veremos el modelo para nuestra y de prueba.

In [18]:
```
# Predicimos el salario con nuestro modelo de regresión lineal.
y_pred = model.predict(x1_test)
```

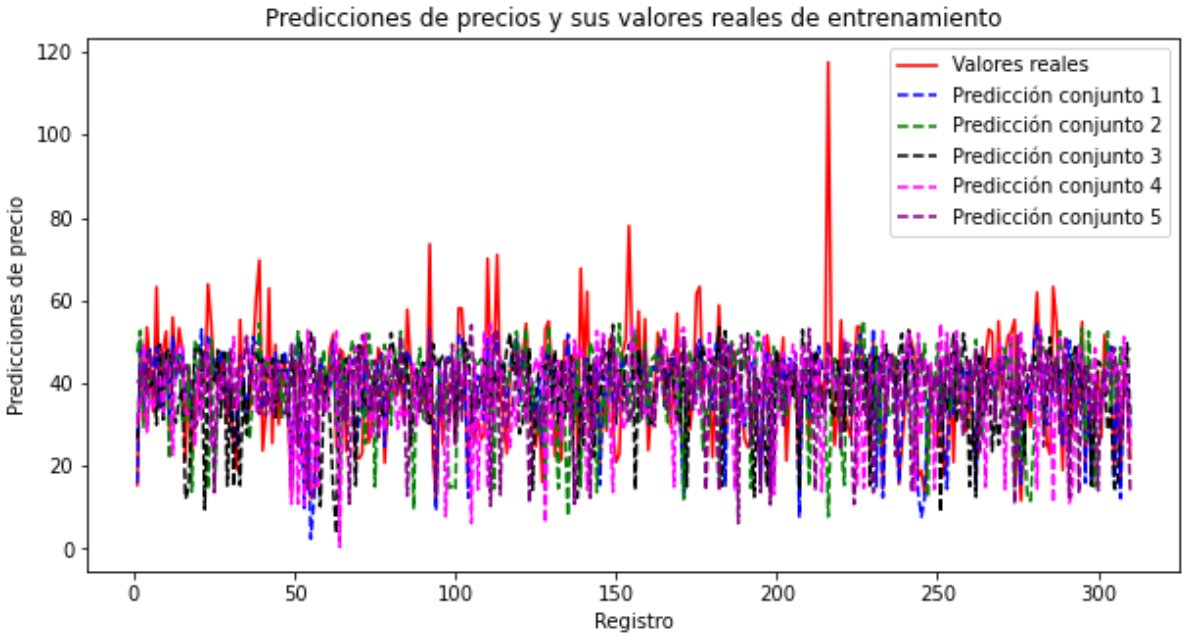
In [19]:
```
# Semillas a utilizar
seeds = [42, 200, 700, 1000]
predictions = []
```

In [20]:
```
for i in range(len(seeds)):
    x_train,x_test,y_train,y_test = train_test_split(df_x, df_y, test_size =
    model = LinearRegression()
    model.fit(x_train, y_train)

    #Agregar predicciones sobre el modelo de entrenamiento
    predictions.append(model.predict(x_train))

    mse = mean_squared_error(predictions[i], y_train)
    mae = mean_absolute_error(predictions[i], y_train)
    r2 = r2_score(predictions[i], y_train)

    print('Resultados de las predicciones con el set de entrenamiento:')
    print('MSE con el conjunto de entrenamiento',i,":",mse)
    print('MAE con el conjunto de entrenamiento',i,":", mae)
    print('El coeficiente R^2:', r2)
    print('**********************************************************')
```

```
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 0 : 83.6886476986626
MAE con el conjunto de entrenamiento 0 : 6.340965742559199
El coeficiente R^2: 0.22067436791606654
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 1 : 84.33375736914839
MAE con el conjunto de entrenamiento 1 : 6.18279466583947
El coeficiente R^2: 0.17708634905395793
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 2 : 80.92079636602112
MAE con el conjunto de entrenamiento 2 : 6.150636059689217
El coeficiente R^2: 0.29498503897655237
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 3 : 85.15780243295286
MAE con el conjunto de entrenamiento 3 : 6.207145121368476
El coeficiente R^2: 0.1464438247005363
**************************************************************
```

La varianza lo medimos observando la diferencia de los errores entre cada modelo. Cada una presenta errores considerables y todas las métricas obtenidas son similares entre ellas. La gráfica siguiente visualiza este comportamiento justamente, las predicciones del modelo se desvian bastante del valor real. Es decir, nuestro modelo de regresión lineal no es adecuado para estimar el precio por unidad de área.

In [21]:
```
xlab = np.arange(1,311)
plt.figure(figsize = (10,5))
plt.plot(xlab, y1_train, '-r',  label = 'Valores reales')
plt.plot(xlab, y_pred_train, '--b', label = 'Predicción conjunto 1')
plt.plot(xlab, predictions[0], '--g', label = 'Predicción conjunto 2')
plt.plot(xlab, predictions[1], linestyle = '--',c = 'black', label = 'Predic
plt.plot(xlab, predictions[2], linestyle = '--', c = 'magenta', label = 'Pre
plt.plot(xlab, predictions[3], linestyle = '--',c = 'purple', label = 'Predi
plt.legend()
plt.title('Predicciones de precios y sus valores reales de entrenamiento')
plt.xlabel('Registro')
plt.ylabel('Predicciones de precio')
plt.show()
```



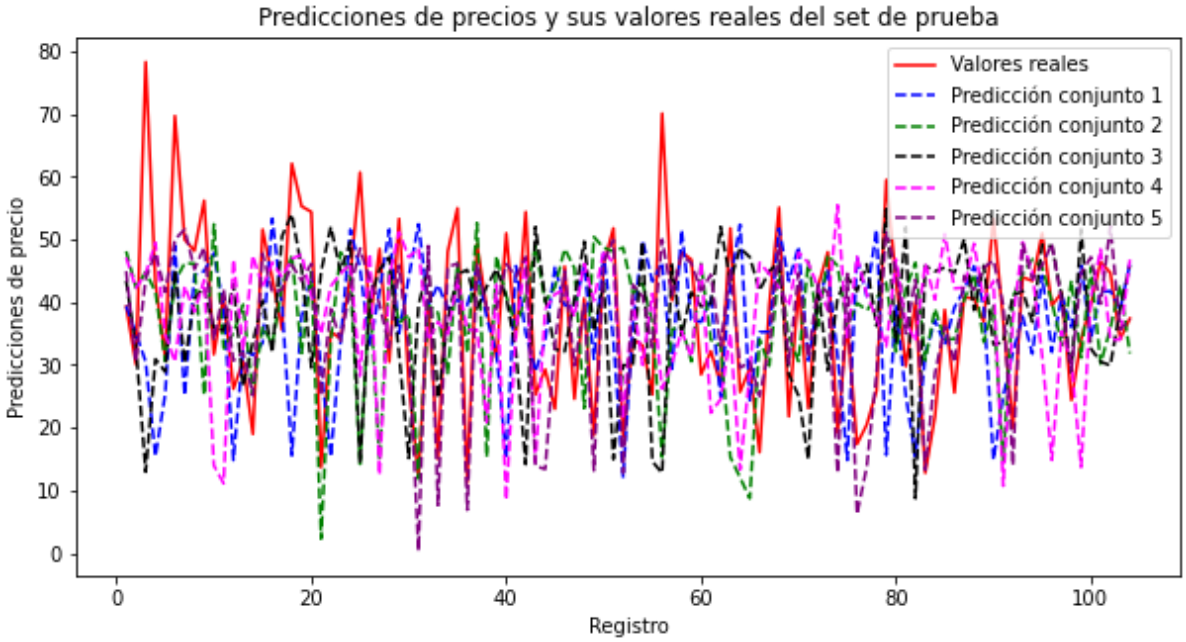
In [22]:
```
seeds = [1800, 42, 200, 700, 1000]
predictions = []
```

In [23]:
```
for i in range(len(seeds)):
    x_train,x_test,y_train,y_test = train_test_split(df_x, df_y, test_size =
    model = LinearRegression()
    model.fit(x_train, y_train)

    #Agregar predicciones sobre el modelo de entrenamiento
    predictions.append(model.predict(x_test))

    mse = mean_squared_error(predictions[i], y_test)
    mae = mean_absolute_error(predictions[i], y_test)
    r2 = r2_score(predictions[i], y_test)

    print('Resultados de las predicciones con el set de entrenamiento:')
    print('MSE con el conjunto de entrenamiento',i,":",mse)
    print('MAE con el conjunto de entrenamiento',i,":", mae)
    print('El coeficiente R^2:', r2)
    print('**********************************************************')
```

```
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 0 : 68.58574815024359
MAE con el conjunto de entrenamiento 0 : 6.0147352156011955
El coeficiente R^2: 0.31353738958106836
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 1 : 66.74857426921457
MAE con el conjunto de entrenamiento 1 : 5.924852914133909
El coeficiente R^2: 0.3667600285259186
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 2 : 64.80892867387837
MAE con el conjunto de entrenamiento 2 : 6.309028874770963
El coeficiente R^2: 0.3920508899194496
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 3 : 76.85930578561712
MAE con el conjunto de entrenamiento 3 : 6.370866980128719
El coeficiente R^2: 0.28833939357502236
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 4 : 63.18595054994734
MAE con el conjunto de entrenamiento 4 : 6.153618252180063
El coeficiente R^2: 0.5436673831223586
**************************************************************
```

In [24]:

```
xlab = np.arange(1,len(y_test)+1)
plt.figure(figsize = (10,5))
plt.plot(xlab, y_test, '-r',  label = 'Valores reales')
plt.plot(xlab, predictions[0], '--b', label = 'Predicción conjunto 1')
plt.plot(xlab, predictions[1], '--g', label = 'Predicción conjunto 2')
plt.plot(xlab, predictions[2], linestyle = '--',c = 'black', label = 'Predic
plt.plot(xlab, predictions[3], linestyle = '--', c = 'magenta', label = 'Pre
plt.plot(xlab, predictions[4], linestyle = '--',c = 'purple', label = 'Predi
plt.legend()
plt.title('Predicciones de precios y sus valores reales del set de prueba')
plt.xlabel('Registro')
plt.ylabel('Predicciones de precio')
plt.show()
```



Por lo tanto, concluimos que nuestro modelo de regresión lineal posee alto sesgo y varianza lo cual se traduce en un modelo inferior. Se recomienda retomar los valores de entrada para obtener un modelo que se ajusta más a los valores reales.

Mejorar el modelo

Dado que nuestro modelo de regresión lineal no fue adecuado, realizamos otro modelo de regresión.

In [25]:
```
col = ['fecha','edad casa','metro','tiendas','lat','long','price']
df = pd.read_csv('Real Estate.csv',index_col = 'No')
df.columns = col
```

In [26]:
```
df_x = df.drop(['price'], axis=1)
```

In [27]:
```
df_y = df['price']
testp = 0.25
seed1 = 1800

x_train,x_test,y_train,y_test = train_test_split(df_x, df_y, test_size = tes
```

Muchas veces cuando trabajamos con un set de datos y usamos un modelo de aprendizaje no sabemos qué hiperparámetros nos dará el mejor resultado. Para obtener el set de mejores hiperparámetros realizaremos un Gridsearch de SciKit Learn donde pasa todas las combinaciones de hiperparámetros uno por uno al modelo hasta que verifica el mejor resultado. En este caso, dado que nuestro modelo de regresión lineal múltiple no era un buen modelo se decidió implementar esta técnica.

In [28]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
```

Aquí utilizamos GradientBoostingRegressor como un modelo de aprendizaje de máquina para utilizar GridSearch.

In [29]:
```
GBR = GradientBoostingRegressor()
```

In [30]:
```
parameters = {'learning_rate': [0.01,0.02,0.03,0.04],
                  'subsample'    : [0.9, 0.5, 0.2, 0.1],
                  'n_estimators' : [100,500,1000, 1500],
                  'max_depth'    : [4,6,8,10]
                 }
```

Ajustamos el GradientBoostingRegressor a nuestros datos de entrenamiento.

In [31]:
```
grid_GBR = GridSearchCV(estimator=GBR, param_grid = parameters, cv = 2, n_jo
grid_GBR.fit(x_train, y_train)
```

Out[31]:
```
GridSearchCV(cv=2, estimator=GradientBoostingRegressor(), n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.02, 0.03, 0.04],
                         'max_depth': [4, 6, 8, 10],
                         'n_estimators': [100, 500, 1000, 1500],
                         'subsample': [0.9, 0.5, 0.2, 0.1]})
```

In [32]:
```
print(" Results from Grid Search " )
print("\n The best estimator across ALL searched params:\n",grid_GBR.best_es
print("\n The best score across ALL searched params:\n",grid_GBR.best_score_
print("\n The best parameters across ALL searched params:\n",grid_GBR.best_p
```

```
 Results from Grid Search

 The best estimator across ALL searched params:
 GradientBoostingRegressor(learning_rate=0.01, max_depth=8, n_estimators=50
0,
                          subsample=0.1)

 The best score across ALL searched params:
 0.6848461766466303

 The best parameters across ALL searched params:
 {'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 500, 'subsample':
0.1}
```

Dado este resultado nos damos cuenta que debemos escoger otro modelo de regresión ya que regresión lineal no es adecuado para nuestra base de datos. Se decidió utilizar un RandomForestRegressor para esto mismo e implementar el hiperparámetro de n_estimators: 500. Establecemos las mismas condiciones que utilizamos para el modelo anterior.

In [33]:
```
testp = 0.25
seed1 = 1800

x_train,x_test,y_train,y_test = train_test_split(df_x, df_y, test_size = tes
```

In [34]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [35]:
```
regressor = RandomForestRegressor(n_estimators = 500, random_state = 1800)
```

In [36]:
```
regressor.fit(x_train, y_train)
```

Out[36]:
```
RandomForestRegressor(n_estimators=500, random_state=1800)
```

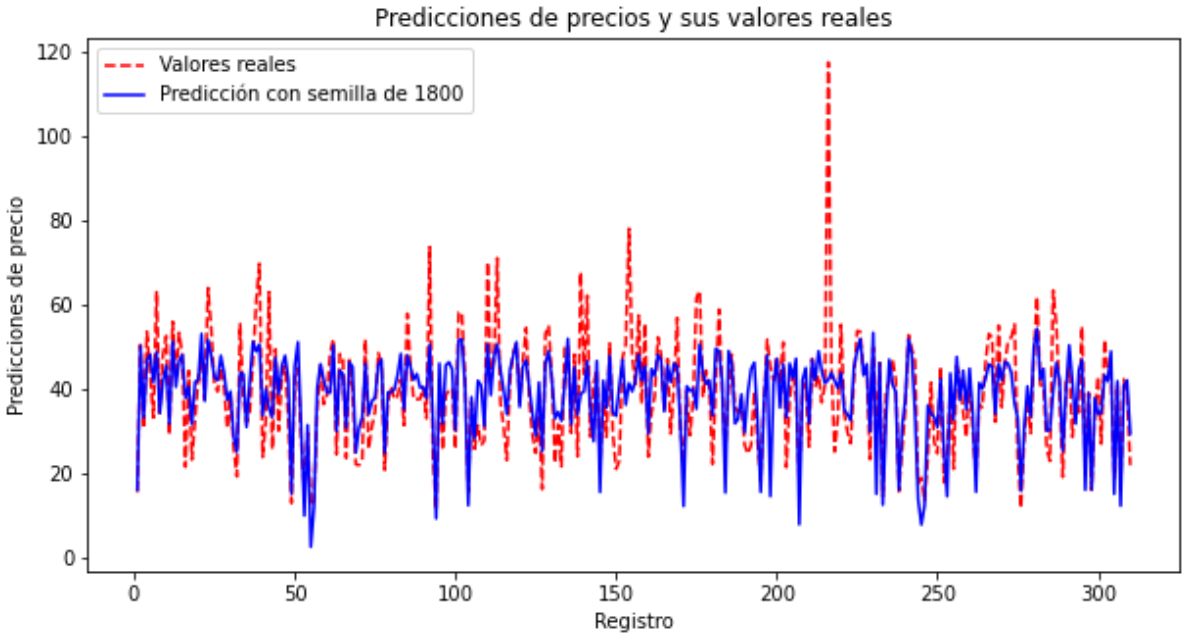
In [37]:
```
y_pred = regressor.predict(x_train)
```

In [38]:
```
mse = mean_squared_error(y_pred, y_train)
mae = mean_absolute_error(y_pred, y_train)
r2 = r2_score(y_pred, y_train)

print('Resultados de las predicciones con el set de entrenamiento:')
print('MSE con el conjunto de entrenamiento:', mse)
print('MAE con el conjunto de entrenamiento:', mae)
print('El coeficiente R^2:', r2)
```

```
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento: 8.847327970220919
MAE con el conjunto de entrenamiento: 1.8588879062980037
El coeficiente R^2: 0.9408294375024103
```

Observamos que al entrenar nuestro regresor y después realizar predicciones sobre este mismo set los errores son muy bajos en comparación al modelo previo. Presentamos un coeficiente de determinación r^2 cercano a 1 el cual indica que la forma del modelo es bueno. En la siguiente grafica tenemos la visualización de esto mismo donde presentamos el valor real del set de entrenamiento y la predicción de este conjunto. Como podemos observar el sesgo es considerablemente bajo ya que las curvas se parecen.

In [39]:
```
xlab = np.arange(1,311)
plt.figure(figsize = (10,5))
plt.plot(xlab, y1_train, '--r',  label = 'Valores reales')
plt.plot(xlab, y_pred_train, '-b', label = 'Predicción con semilla de 1800')
plt.legend()
plt.title('Predicciones de precios y sus valores reales')
plt.xlabel('Registro')
plt.ylabel('Predicciones de precio')
plt.show()
```



Para evaluar la varianza implementamos un nuevo modelo con otro conjunto de datos. Para esto, modificamos la semilla al dividir el set de entrenamiento para observar la variación entre los modelos.

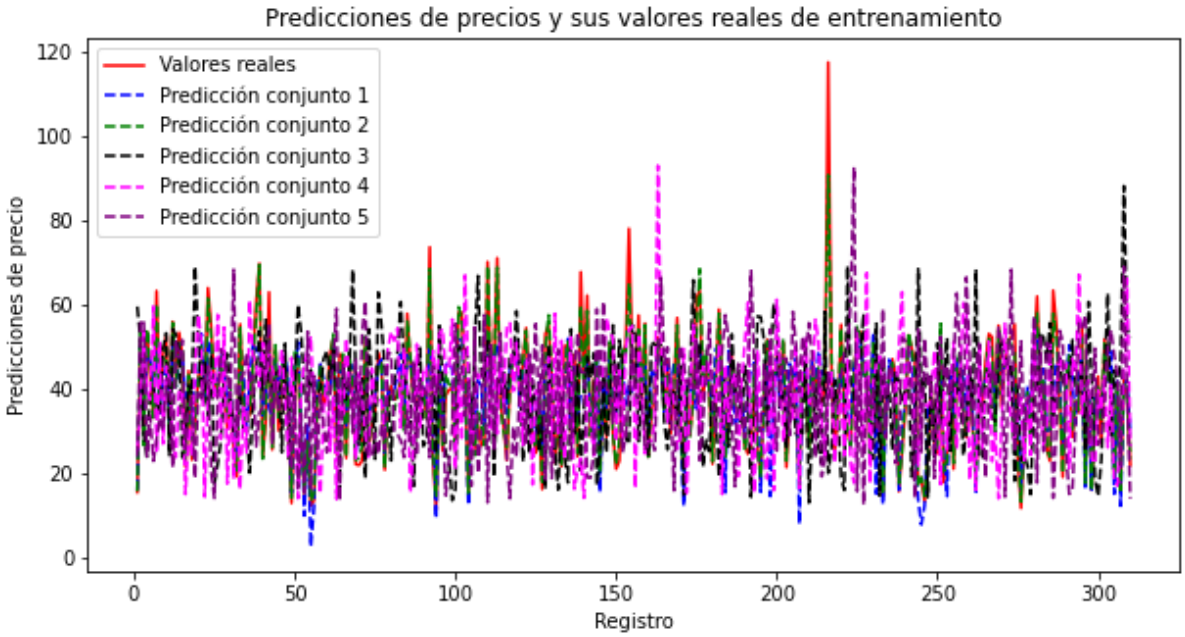
In [40]:
```
seeds = [1800, 42, 200, 700, 1000]
predictions = []
```

In [41]:
```
for i in range(len(seeds)):
    x_train,x_test,y_train,y_test = train_test_split(df_x, df_y, test_size =
    regressor = RandomForestRegressor(n_estimators = 500, random_state = 180
    regressor.fit(x_train, y_train)
    #Agregar predicciones sobre el modelo de entrenamiento
    predictions.append(regressor.predict(x_train))

    mse = mean_squared_error(predictions[i], y_train)
    mae = mean_absolute_error(predictions[i], y_train)
    r2 = r2_score(predictions[i], y_train)

    print('Resultados de las predicciones con el set de entrenamiento:')
    print('MSE con el conjunto de entrenamiento',i,":", mse)
    print('MAE con el conjunto de entrenamiento',i,":", mae)
    print('El coeficiente R^2:', r2)
    print('**********************************************************')
```

```
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 0 : 8.881926678754132
MAE con el conjunto de entrenamiento 0 : 1.9134874180747583
El coeficiente R^2: 0.9419737412280298
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 1 : 9.086516973361261
MAE con el conjunto de entrenamiento 1 : 1.8820924884792658
El coeficiente R^2: 0.9435011901705118
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 2 : 7.935173369099379
MAE con el conjunto de entrenamiento 2 : 1.7920631075268763
El coeficiente R^2: 0.9496813764890778
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 3 : 8.137678072324299
MAE con el conjunto de entrenamiento 3 : 1.736738815668208
El coeficiente R^2: 0.9514177577696784
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 4 : 8.975453121128757
MAE con el conjunto de entrenamiento 4 : 1.8545260153609766
El coeficiente R^2: 0.9417754742621232
**************************************************************
```

Observamos que para la varianza, la diferencia entre los errores entre los modelos es mínima, es decir, las métricas obtenidas se asimilan. En la siguiente gráfica observamos este comportamiento ya que la predicciones obtenidas y curvas son parecidas.

In [42]:
```
xlab = np.arange(1,311)
plt.figure(figsize = (10,5))
plt.plot(xlab, y1_train, '-r',  label = 'Valores reales')
plt.plot(xlab, y_pred_train, '--b', label = 'Predicción conjunto 1')
plt.plot(xlab, predictions[0], '--g', label = 'Predicción conjunto 2')
plt.plot(xlab, predictions[1], linestyle = '--',c = 'black', label = 'Predic
plt.plot(xlab, predictions[2], linestyle = '--', c = 'magenta', label = 'Pre
plt.plot(xlab, predictions[3], linestyle = '--',c = 'purple', label = 'Predi
plt.legend()
plt.title('Predicciones de precios y sus valores reales de entrenamiento')
plt.xlabel('Registro')
plt.ylabel('Predicciones de precio')
plt.show()
```



En conclusión, el modelo posee sesgo bajo con varianza baja un indicador de que construimos un buen modelo. Sin embargo, puede haber un sobreajuste al momento de obtener las predicciones con el set de entrenamiento.

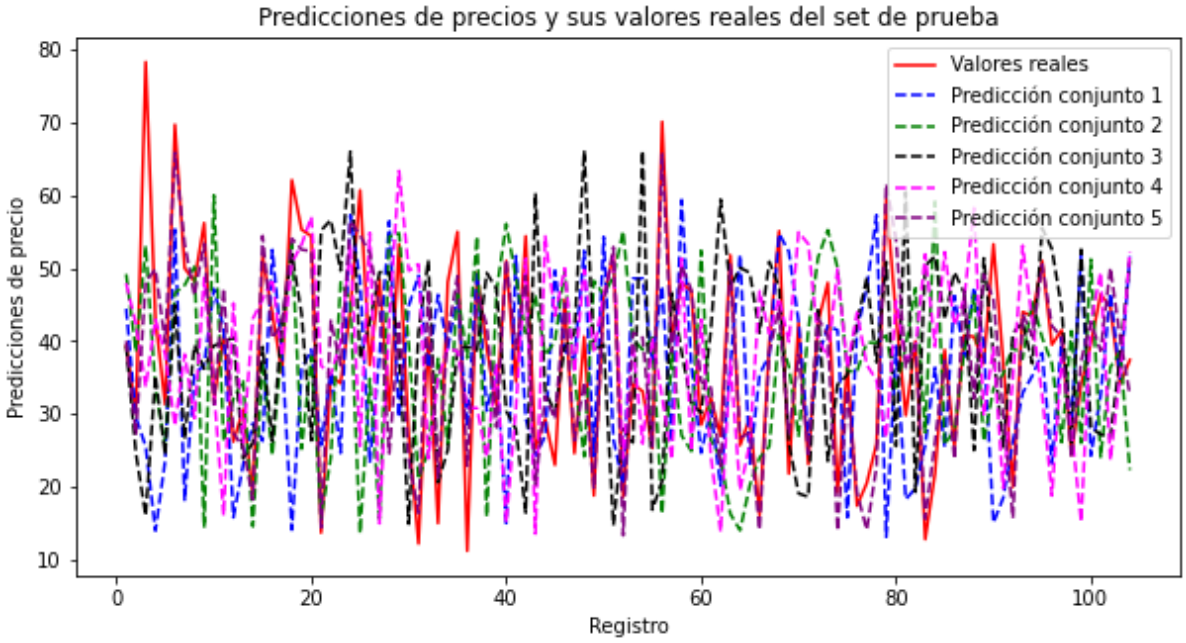
In [43]:
```
seeds = [1800, 42, 200, 700, 1000]
predictions = []
```

In [44]:
```
for i in range(len(seeds)):
    x_train,x_test,y_train,y_test = train_test_split(df_x, df_y, test_size =
    regressor = RandomForestRegressor(n_estimators = 500, random_state = 180
    regressor.fit(x_train, y_train)
    #Agregar predicciones sobre el modelo de entrenamiento
    predictions.append(regressor.predict(x_test))

    mse = mean_squared_error(predictions[i], y_test)
    mae = mean_absolute_error(predictions[i], y_test)
    r2 = r2_score(predictions[i], y_test)

    print('Resultados de las predicciones con el set de entrenamiento:')
    print('MSE con el conjunto de entrenamiento',i,":",mse)
    print('MAE con el conjunto de entrenamiento',i,":", mae)
    print('El coeficiente R^2:', r2)
    print('**********************************************************')
```

```
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 0 : 42.14392311764837
MAE con el conjunto de entrenamiento 0 : 4.302222798382181
El coeficiente R^2: 0.7042994419423734
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 1 : 41.23708844694416
MAE con el conjunto de entrenamiento 1 : 4.24424657738094
El coeficiente R^2: 0.7028857450715209
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 2 : 39.69790899165058
MAE con el conjunto de entrenamiento 2 : 4.762934670329683
El coeficiente R^2: 0.737600727210937
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 3 : 49.65597261956567
MAE con el conjunto de entrenamiento 3 : 5.05736427655677
El coeficiente R^2: 0.6133353398002499
**************************************************************
Resultados de las predicciones con el set de entrenamiento:
MSE con el conjunto de entrenamiento 4 : 36.146302411427705
MAE con el conjunto de entrenamiento 4 : 4.485294505494512
El coeficiente R^2: 0.7713107101416914
**************************************************************
```

In [45]:

```
xlab = np.arange(1,len(y_test)+1)
plt.figure(figsize = (10,5))
plt.plot(xlab, y_test, '-r',  label = 'Valores reales')
plt.plot(xlab, predictions[0], '--b', label = 'Predicción conjunto 1')
plt.plot(xlab, predictions[1], '--g', label = 'Predicción conjunto 2')
plt.plot(xlab, predictions[2], linestyle = '--',c = 'black', label = 'Predic
plt.plot(xlab, predictions[3], linestyle = '--', c = 'magenta', label = 'Pre
plt.plot(xlab, predictions[4], linestyle = '--',c = 'purple', label = 'Predi
plt.legend()
plt.title('Predicciones de precios y sus valores reales del set de prueba')
plt.xlabel('Registro')
plt.ylabel('Predicciones de precio')
plt.show()
```



Para nuestro último modelo observamos una disminución en la calidad del modelo ya que los valores de MSE y MAE aumentan y r^2 disminuye. La visualización de las gráficas de arriba confirman este comportamiento ya que la distintas entre los valores reales y las predicciones son mayor.

Gracias a los últimos datos obtenidos podemos analizar mejor el sesgo y varianza. De manera visual, conocemos el sesgo ya que observamos que las curvas de predicción se asimilan al conjunto de datos reales de entrenamiento. No obstante, al implementar el modelo utilizando el set de datos de prueba observamos un alejamiento a los valores reales. Es decir, hay sesgo en el modelo lo cual indica que nuestro modelo posee un sobreajuste.

La varianza se modifica con la última implementación del modelo, es decir, las curvas de predicción se desvían más del la curva de valores reales como vemos con la métrica de r^2. Tener un alto sesgo está relacionado con un sobreajuste de los datos.

En conclusión, nuestro nuevo modelo presenta un sesgo moderado con varianza moderada. Por lo tanto, se declara que el modelo presenta un sobreajuste.