ModeloViviendaPrecioFEVM

February 5, 2024

1 Quiero comprar una casa: Predecir precios de viviendas

Contexto: Estamos buscando comprar una vivienda por la zona de Ames, Iowa. Eres adicto a los chollos y quieres comprar a muy buen precio.

Objetivo: De alguna manera, ganar confianza sobre los precios de las viviendas para poder comprar al mejor precio.

Metodología: Crear un modelo de regresión para predecir los precios de las viviendas.

Datos: Datos históricos de ventas de viviendas en la zona de Ames, Iowa. Ver `data_description.txt` para más contexto.

1.1 Que tienes que hacer tu?

Como mínimo, seguir las instrucciones de este notebook y contestar a las preguntas para conseguir crear un modelo que cumple con el objetivo del proyecto. Las preguntas cubren los requisitos mínimos para poder crear un modelo, pero es recomendable ir un poco más allá para aprender más.

1.1.1 Entregable: cómo hacerlo y qué incluir

Lo más sencillo es copiar este notebook y trabajar directamente dentro de ello.

Antes de todo *no hay ninguna respuesta "correcta"*. Lo importante es tomar decisiones y razonar estas decisiones. Este razonamiento se debe basar en análisis de los datos y tu conocimiento del problema.

El contenido que incluyáis debería

1. Contestar a las preguntas

- Código es parte de contestar a las preguntas
- Es necesario incluir texto para contestar a preguntas (castellano o ingles)
- Gráficos ayudan a explicar tus argumentos

2. Ser auto-explicativo

- El código que genera un análisis debería estar cerca del texto de este análisis - escribe como si estuvieras contando un cuento
- Lo más importante es explicar tu razonamiento en cada paso
- El notebook se debería de poder ejecutar de arriba abajo

3. Ser *breve* - si algo no contribuye a contestar a las preguntas, por favor no incluirlo

- A veces uno intenta hacer algo que no funciona - esto también ayuda a contestar la pregunta (para saber lo que NO funciona)
- Si haces una algo que te parece muy interesante pero no ayuda mucho, podéis incluirlo en un fichero aparte

Es muy normal que como vas avanzando tus respuestas a preguntas anteriores pueden cambiar. Si has contestado algo y luego te has dado cuenta que no te guste tu respuesta - cámbialo! No hay problema. En realidad esto es un proceso cíclico, no lineal.

No hay requisitos de librerías / lenguajes pero es altamente recomendable usar Python y tirar principalmente de `Pandas` y `scikit-learn`.

Por qué lo hacemos asi? Este formato es algo muy típico de las pruebas que hacen las empresas en procesos de selección, donde el objetivo es mostrar tus habilidades y como te acercas a problemas reales de data science. Intentamos replicar esto para que podéis ganar confianza al futuro si os enfrentéis a esto.

1.1.2 Estructura

El notebook se compone por varias secciones que reflejan los pasos típicos de la creación de un modelo. Iremos introduciendo estas secciones en diferentes semanas. Cada sección depende de lo que has hecho anteriormente.

Cada sección viene con las preguntas de la sección. Lo más normal para contestar a la pregunta seria

1. Empezar con la pregunta - dejando tus pensamientos iniciales
2. Código y texto que ayuda en desarrollar tu respuesta
3. Un comentario final que expone tu "respuesta" definitiva - un resumen de los comentarios anteriores

Nota que a veces la "pregunta" es más bien una instrucción. Aquí esperamos un un resultado final más que una respuesta en texto. Aún así, es importante incluir explicación de lo que estás haciendo!

Los datos se encuentran en `primer-ejercicio/data/house-price-data.csv`.

1.2 Entender los datos

Lo más importante de cualquier problema de data science es entender los datos. Si no entiendes - no puedes crear un modelo que tenga sentido.

1.2.1 Cargar los datos

Lo primero de todo es cargar los datos, asegurar que está todo bien y empezar a formar ideas sobre como se relacionan los datos con nuestro problema.

1. Cargar los datos e imprimir las primeras 5 filas

[1]:
```
from pathlib import Path
import pandas as pd
import plotnine as pn
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import numpy as np
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score

pd.set_option('display.max_rows', None) # Para poder ver el max de filas
pd.set_option('display.max_columns', None) # Para poder ver el max de columnas

df = pd.read_csv("house-price-data.csv")

# Imprime las primeras 5 líneas
df.head()
```

[1]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
0	1	60	RL	65.0	8450	Pave	NaN	Reg
1	2	20	RL	80.0	9600	Pave	NaN	Reg
2	3	60	RL	68.0	11250	Pave	NaN	IR1
3	4	70	RL	60.0	9550	Pave	NaN	IR1
4	5	60	RL	84.0	14260	Pave	NaN	IR1

	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1
0	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
1	Lvl	AllPub	FR2	Gtl	Veenker	Feedr
2	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
3	Lvl	AllPub	Corner	Gtl	Crawfor	Norm
4	Lvl	AllPub	FR2	Gtl	NoRidge	Norm

	Condition2	BldgType	HouseStyle	OverallQual	OverallCond	YearBuilt
0	Norm	1Fam	2Story	7	5	2003
1	Norm	1Fam	1Story	6	8	1976
2	Norm	1Fam	2Story	7	5	2001
3	Norm	1Fam	2Story	7	5	1915
4	Norm	1Fam	2Story	8	5	2000

	YearRemodAdd	RoofStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType
0	2003	Gable	CompShg	VinylSd	VinylSd	BrkFace
1	1976	Gable	CompShg	MetalSd	MetalSd	NaN
2	2002	Gable	CompShg	VinylSd	VinylSd	BrkFace
3	1970	Gable	CompShg	Wd Sdng	Wd Shng	NaN
4	2000	Gable	CompShg	VinylSd	VinylSd	BrkFace

	MasVnrArea	ExterQual	ExterCond	Foundation	BsmtQual	BsmtCond	BsmtExposure
0	196.0	Gd	TA	PConc	Gd	TA	No
1	0.0	TA	TA	CBlock	Gd	TA	Gd

```
2       162.0        Gd        TA      PConc       Gd       TA           Mn
3         0.0        TA        TA     BrkTil       TA       Gd           No
4       350.0        Gd        TA      PConc       Gd       TA           Av

  BsmtFinType1  BsmtFinSF1 BsmtFinType2  BsmtFinSF2  BsmtUnfSF  TotalBsmtSF  \
0          GLQ         706          Unf           0        150          856
1          ALQ         978          Unf           0        284         1262
2          GLQ         486          Unf           0        434          920
3          ALQ         216          Unf           0        540          756
4          GLQ         655          Unf           0        490         1145

  Heating HeatingQC CentralAir Electrical  1stFlrSF  2ndFlrSF  LowQualFinSF  \
0    GasA        Ex          Y      SBrkr       856       854             0
1    GasA        Ex          Y      SBrkr      1262         0             0
2    GasA        Ex          Y      SBrkr       920       866             0
3    GasA        Gd          Y      SBrkr       961       756             0
4    GasA        Ex          Y      SBrkr      1145      1053             0

   GrLivArea  BsmtFullBath  BsmtHalfBath  FullBath  HalfBath  BedroomAbvGr  \
0       1710             1             0         2         1             3
1       1262             0             1         2         0             3
2       1786             1             0         2         1             3
3       1717             1             0         1         0             3
4       2198             1             0         2         1             4

   KitchenAbvGr KitchenQual  TotRmsAbvGrd Functional  Fireplaces FireplaceQu  \
0             1          Gd             8        Typ           0         NaN
1             1          TA             6        Typ           1          TA
2             1          Gd             6        Typ           1          TA
3             1          Gd             7        Typ           1          Gd
4             1          Gd             9        Typ           1          TA

  GarageType  GarageYrBlt GarageFinish  GarageCars  GarageArea GarageQual  \
0     Attchd       2003.0          RFn           2         548         TA
1     Attchd       1976.0          RFn           2         460         TA
2     Attchd       2001.0          RFn           2         608         TA
3     Detchd       1998.0          Unf           3         642         TA
4     Attchd       2000.0          RFn           3         836         TA

  GarageCond PavedDrive  WoodDeckSF  OpenPorchSF  EnclosedPorch  3SsnPorch  \
0         TA          Y           0           61              0          0
1         TA          Y         298            0              0          0
2         TA          Y           0           42              0          0
3         TA          Y           0           35            272          0
4         TA          Y         192           84              0          0

   ScreenPorch  PoolArea PoolQC Fence MiscFeature  MiscVal  MoSold  YrSold  \
```

```
0           0         0    NaN   NaN         NaN        0       2    2008
1           0         0    NaN   NaN         NaN        0       5    2007
2           0         0    NaN   NaN         NaN        0       9    2008
3           0         0    NaN   NaN         NaN        0       2    2006
4           0         0    NaN   NaN         NaN        0      12    2008

  SaleType SaleCondition  SalePrice
0       WD        Normal     208500
1       WD        Normal     181500
2       WD        Normal     223500
3       WD       Abnorml     140000
4       WD        Normal     250000
```

2. Se han cargado de forma correcta los datos? Cómo lo sabemos?

[2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     588 non-null    object
```

```
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
59  GarageYrBlt    1379 non-null   float64
60  GarageFinish   1379 non-null   object
61  GarageCars     1460 non-null   int64
62  GarageArea     1460 non-null   int64
63  GarageQual     1379 non-null   object
64  GarageCond     1379 non-null   object
65  PavedDrive     1460 non-null   object
66  WoodDeckSF     1460 non-null   int64
67  OpenPorchSF    1460 non-null   int64
68  EnclosedPorch  1460 non-null   int64
69  3SsnPorch      1460 non-null   int64
70  ScreenPorch    1460 non-null   int64
71  PoolArea       1460 non-null   int64
72  PoolQC         7 non-null      object
73  Fence          281 non-null    object
```

```
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

Podemos comprobar con df.info() que ha cargado el total de filas que tenemos en el csv, 1460. Tambien se puede ver que tenemos 81 columnas. Aunque alguno de las columnas se puede ver que tienen muchos nulls como por ejemplo "Alley". Y en el tipo de dato encajaria con los tipos numericos y texto que tenemos en el csv.

3. Tras una mirada inicial - que parece que tenemos en nuestros datos?

[3]:
```
df.nunique()
```

[3]:
```
Id               1460
MSSubClass         15
MSZoning            5
LotFrontage       110
LotArea          1073
Street              2
Alley               2
LotShape            4
LandContour         4
Utilities           2
LotConfig           5
LandSlope           3
Neighborhood       25
Condition1          9
Condition2          8
BldgType            5
HouseStyle          8
OverallQual        10
OverallCond         9
YearBuilt         112
YearRemodAdd       61
RoofStyle           6
RoofMatl            8
Exterior1st        15
Exterior2nd        16
MasVnrType          3
MasVnrArea        327
ExterQual           4
ExterCond           5
```

Foundation	6
BsmtQual	4
BsmtCond	4
BsmtExposure	4
BsmtFinType1	6
BsmtFinSF1	637
BsmtFinType2	6
BsmtFinSF2	144
BsmtUnfSF	780
TotalBsmtSF	721
Heating	6
HeatingQC	5
CentralAir	2
Electrical	5
1stFlrSF	753
2ndFlrSF	417
LowQualFinSF	24
GrLivArea	861
BsmtFullBath	4
BsmtHalfBath	3
FullBath	4
HalfBath	3
BedroomAbvGr	8
KitchenAbvGr	4
KitchenQual	4
TotRmsAbvGrd	12
Functional	7
Fireplaces	4
FireplaceQu	5
GarageType	6
GarageYrBlt	97
GarageFinish	3
GarageCars	5
GarageArea	441
GarageQual	5
GarageCond	5
PavedDrive	3
WoodDeckSF	274
OpenPorchSF	202
EnclosedPorch	120
3SsnPorch	20
ScreenPorch	76
PoolArea	8
PoolQC	3
Fence	4
MiscFeature	4
MiscVal	21

```
MoSold             12
YrSold              5
SaleType            9
SaleCondition       6
SalePrice         663
dtype: int64
```

Podemos comprobar los valores unicos que tienen cada columna y con ayuda del data_description.txt podemos ver que en algunos casos hay mas datos en el data_description que datos reales en el csv. Pero en otros casos tenemos el problema como en PoolQC que los NA que son No Pool lo detecta como un null, vamos a tener que tratar esto mas adelante.

1.2.2 Analizar los datos

Para entender lo que tenemos en los datos, los tenemos que analizar. En realidad, este paso es el más importante de todos y puede durar horas, días o meses (la verdad es que en una empresa, nunca terminamos de analizar los datos). El conocimiento que ganamos aquí forma la base del razonamiento que usaremos para hacer decisiones en el futuro.

Aquí buscamos cosas como

1. El comportamiento de nuestros datos
2. Los problemas que podemos tener
3. Una impresión inicial de lo que podemos y de lo no podemos hacer con estos datos
4. Las metodologías que seguramente vamos a usar

1. Cuantas filas de datos tenemos? Qué representa cada fila?

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

[4]:
```
(1460, 81)
```

Tenemos 1460 filas, cada fila es un registro de cada venta de casa con muchas columnas de informacion.

2. Hay filas que no son relevantes para nuestro problema?

A priori ami juicio creo que todas las filas son utiles, aunque mas adelante la 523 me da problemas he decidido dejarla porque borrarlo seria alterar los datos originales.

3. Qué variables crees que van a ser los más importantes? Como se comportan estos datos? Qué variable será nuestro target?

[5]:
```
# Obtén las columnas categóricas
categorical_columns = df.select_dtypes(include=['object']).columns

# Convierte las columnas categóricas a variables dummy
df2 = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# Calcula la matriz de correlación con todas las variables
correlation_matrix = df2.corr()
```

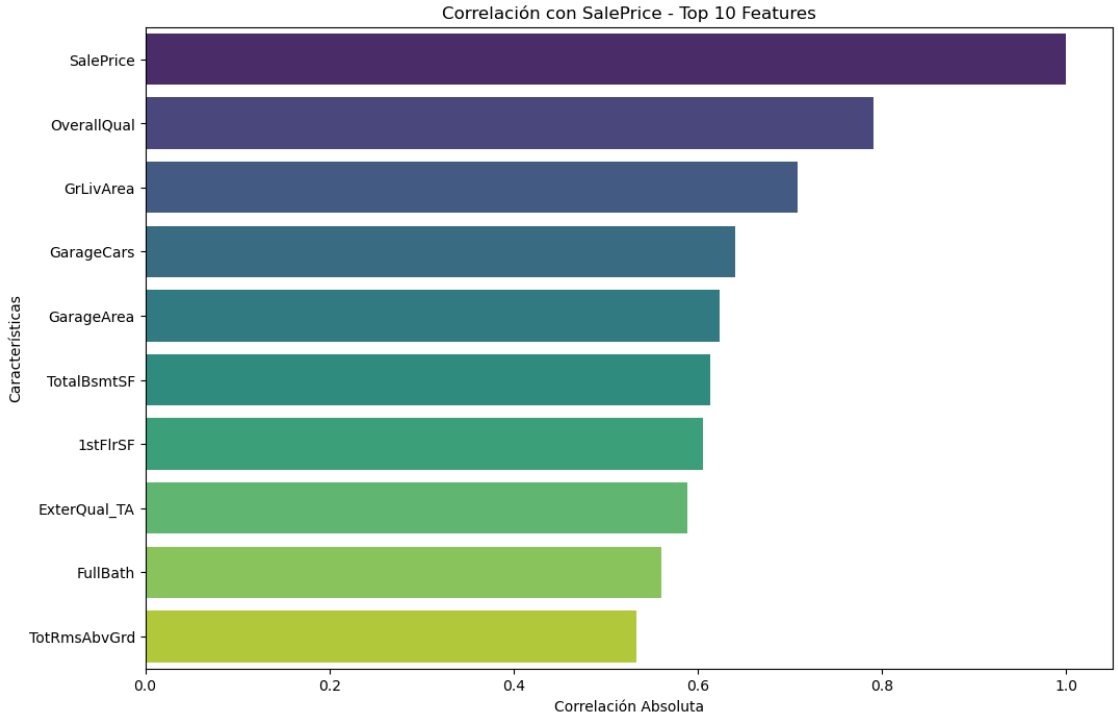
```
saleprice_correlation = correlation_matrix['SalePrice'].abs().
 ↪sort_values(ascending=False)

saleprice_correlation.index
# # Convertir la serie a un DataFrame
best_features_df = pd.DataFrame({'Correlation': saleprice_correlation.index,␣
 ↪'Number': saleprice_correlation.values})
# # Mostrar todas las variables seleccionadas
best_features_df.head(10)
```

[5]:

	Correlation	Number
0	SalePrice	1.000000
1	OverallQual	0.790982
2	GrLivArea	0.708624
3	GarageCars	0.640409
4	GarageArea	0.623431
5	TotalBsmtSF	0.613581
6	1stFlrSF	0.605852
7	ExterQual_TA	0.589044
8	FullBath	0.560664
9	TotRmsAbvGrd	0.533723

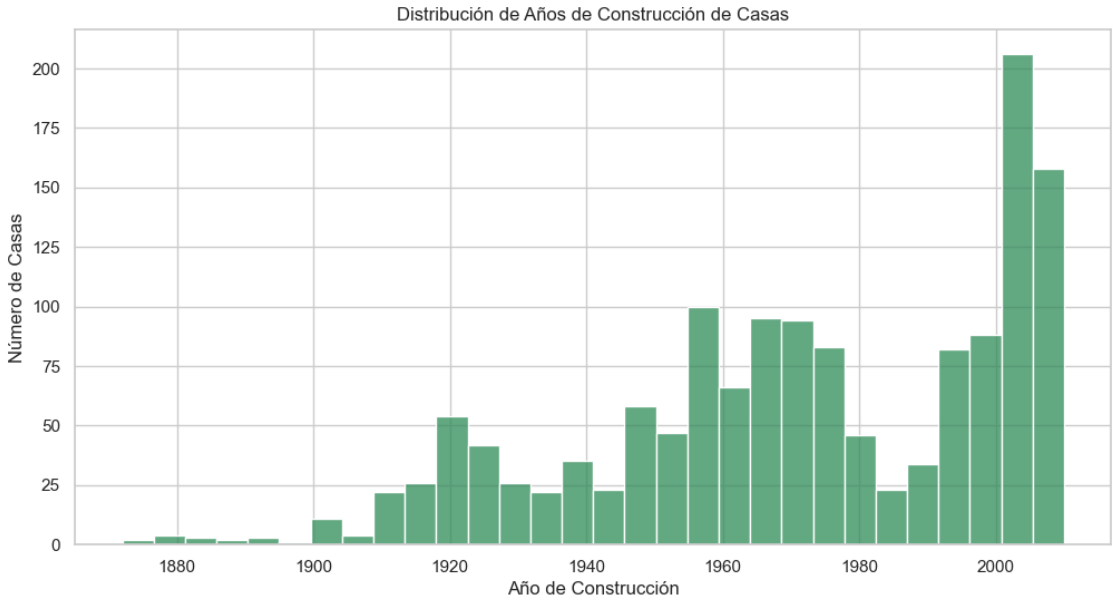
Convertimos todas las columnas de tipo object en columnas dummy, es decir se generan muchas columnas, esto a priori puede hacer que le modelo sea bastante complicado de seguir ya que pasamos de 81 columnas a 246 columnas.

[6]:
```
plt.figure(figsize=(12, 8))
sns.barplot(x=saleprice_correlation.head(10).values, y=saleprice_correlation.
 ↪head(10).index, palette='viridis')
plt.title('Correlación con SalePrice - Top 10 Features')
plt.xlabel('Correlación Absoluta')
plt.ylabel('Características')
plt.show()
```



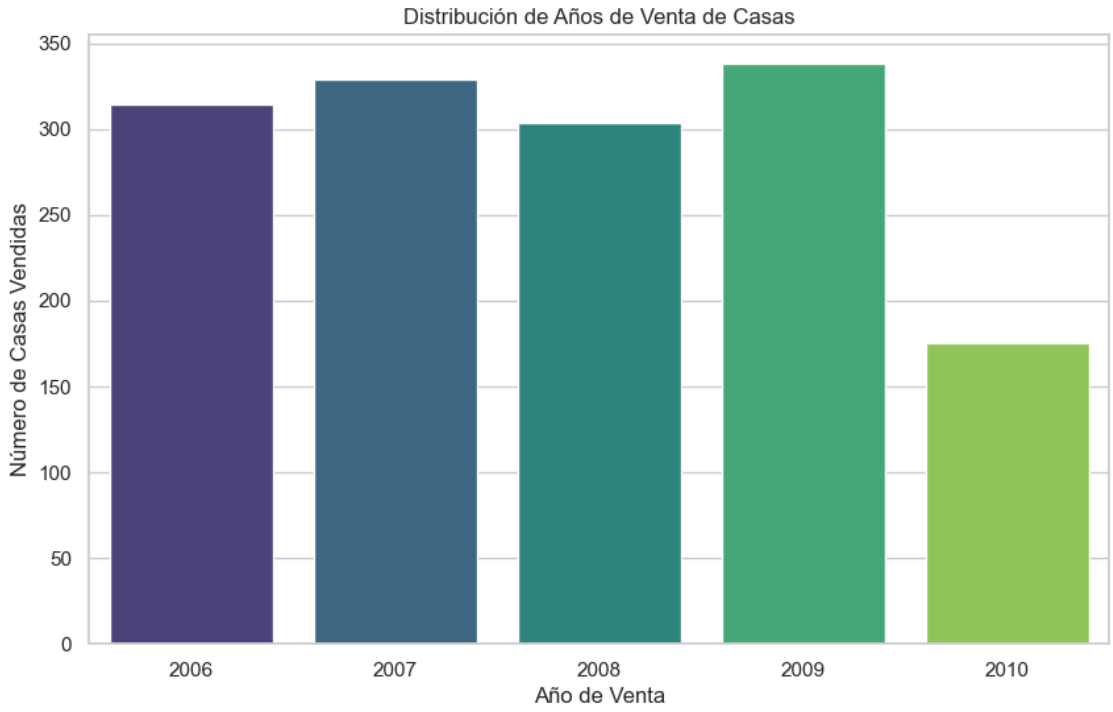
4. Cuándo se construyeron los diferentes casas? Dibuja un gráfico para visualizarlo

[7]:
```
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

sns.histplot(df['YearBuilt'], bins=30, kde=False, color='#2E8B57')

plt.title('Distribución de Años de Construcción de Casas')
plt.xlabel('Año de Construcción')
plt.ylabel('Número de Casas')
plt.show()
```



5. Cuánto tiempo tardaron las diferentes casas en venderse? Dibuja un gráfico para visualizarlo

[8]:
```
sns.set(style='whitegrid', palette='pastel')

plt.figure(figsize=(10, 6))
sns.countplot(x='YrSold', data=df, palette='viridis')
plt.title('Distribución de Años de Venta de Casas')
plt.xlabel('Año de Venta')
plt.ylabel('Número de Casas Vendidas')
plt.show()
```



[9]:
```
print(df.dtypes)
```

```
Id                int64
MSSubClass        int64
MSZoning         object
LotFrontage     float64
LotArea           int64
Street           object
Alley            object
LotShape         object
LandContour      object
Utilities        object
LotConfig        object
LandSlope        object
Neighborhood     object
Condition1       object
Condition2       object
BldgType         object
HouseStyle       object
OverallQual       int64
OverallCond       int64
YearBuilt         int64
YearRemodAdd      int64
RoofStyle        object
```

```
RoofMatl          object
Exterior1st       object
Exterior2nd       object
MasVnrType        object
MasVnrArea       float64
ExterQual         object
ExterCond         object
Foundation        object
BsmtQual          object
BsmtCond          object
BsmtExposure      object
BsmtFinType1      object
BsmtFinSF1         int64
BsmtFinType2      object
BsmtFinSF2         int64
BsmtUnfSF          int64
TotalBsmtSF        int64
Heating           object
HeatingQC         object
CentralAir        object
Electrical        object
1stFlrSF           int64
2ndFlrSF           int64
LowQualFinSF       int64
GrLivArea          int64
BsmtFullBath       int64
BsmtHalfBath       int64
FullBath           int64
HalfBath           int64
BedroomAbvGr       int64
KitchenAbvGr       int64
KitchenQual       object
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars         int64
GarageArea         int64
GarageQual        object
GarageCond        object
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
```

```
ScreenPorch        int64
PoolArea           int64
PoolQC            object
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
dtype: object
```

Ver el tipo de datos que tiene cada columna

[10]:
```
df2.shape
```

[10]:
```
(1460, 246)
```

[11]:
```
best_features_df.iloc[1:10]['Correlation'].tolist()
```

[11]:
```
['OverallQual',
 'GrLivArea',
 'GarageCars',
 'GarageArea',
 'TotalBsmtSF',
 '1stFlrSF',
 'ExterQual_TA',
 'FullBath',
 'TotRmsAbvGrd']
```

Las 10 columnas que mas correlacion tendria con SalePrice, en formato lista

[12]:
```
df.SalePrice.min()
```

[12]:
```
34900
```

1.2.3 Limpiar los datos

Cuando creamos un modelo, no hay forma de "mágicamente" contar al modelo que es lo que queremos. Tenemos que usar unos datos que representan bien nuestro problema y crear un modelo que predice algo que alinea con la respuesta que buscamos.

Si los datos son "sucios" (malos, con problemas, reflejan algo que no representa bien a nuestro problema) - nuestro modelo va a predecir cosas que no acaban de tener mucho sentido, no son de fiar o directamente son equivocadas.

Cosas que queremos evitar

- Datos que no tienen que ver con nuestro problema
- Datos que tienen poca muestra

- Outliers
- Nulos

1. Identifique los principales problemas que tienen las variables que parecen (por ahora) más interesantes para el modelo

Pues datos nulls que afectan al modelo y datos de tipo Object que hay que borrarlos o convertirlos a formato numerico.

2. Arregla los problemas

[13]:
```
# Obtén las columnas que tienen valores nulos y llenar con la media del resto␣
↪de datos de cada columna
columnas_con_nulls = df2.columns[df2.isnull().any()].tolist()

for columna in columnas_con_nulls:
    media_columna = df2[columna].mean()
    df2[columna] = df2[columna].fillna(media_columna)
```

Los datos de tipo Object los converti a columnas dummy mas arriba

3. Borra todos los datos que no son relevantes para el problema - simplifica los datos

En mi caso para lo que quiero probar quiero dejar todos los datos ya que creo que todos pueden servir en menor o mayor medida

1.3 Preparación de los datos para el modelo

Ahora que entendemos bien los datos, tomamos un paso más directo hacia la creación de nuestro modelo preparando los datos para el entrenamiento.

1.3.1 Relevancia de variables

Por norma general, queremos incluir variables que son importantes en la predicción de nuestro target:

- Tener menos variables más relevantes suele ser mejor - porque simplifica el modelo
- Si no tenemos variables relevantes, los resultados del modelo van a ser malos

Para analizar la relevancia, normalmente miramos que haya "correlaciones" entre diferentes variables y el target. Hay muchas formas de analizar estas "correlaciones", como por ejemplo:

- Coeficiente de correlación (variables continuas)
- La media del target para cada valor del variable (variables NO continuas)

1. Qué variables son los más relevantes? Analiza la relevancia para comprobarlo

[14]:
```
# Crear listas para almacenar los resultados
num_features_list = []
r2_list = []
mae_list = []
rmse_list = []
```

```
# Obtén la lista de características
all_features_list = best_features_df.iloc[1:, 0].tolist()

num_features_to_try = len(all_features_list)

for i in range(1, num_features_to_try + 1):
    # Selecciona las primeras i características
    selected_features = all_features_list[:i]

    # Divide los datos en conjunto de entrenamiento y prueba
    X_train, X_test, y_train, y_test = train_test_split(df2[selected_features],␣
 ↪df2.SalePrice, test_size=0.2, random_state=8)

    # Crea y entrena el modelo de regresión lineal
    reg = linear_model.LinearRegression()
    reg.fit(X_train, y_train)

    # Realiza predicciones en el conjunto de prueba
    predictions = reg.predict(X_test)

    # Calcula el coeficiente de determinación (R^2)
    r2 = r2_score(y_test, predictions)

    # Calcula el MAE y el RMSE
    mae = mean_absolute_error(y_test, predictions)
    rmse = np.sqrt(mean_squared_error(y_test, predictions))

    # Almacena los resultados en las listas
    num_features_list.append(i)
    r2_list.append(r2)
    mae_list.append(mae)
    rmse_list.append(rmse)

# Crea el DataFrame a partir de las listas
results_df = pd.DataFrame({'Num_features': num_features_list, 'R2': r2_list,␣
 ↪'MAE': mae_list, 'RMSE': rmse_list})

# Muestra los resultados
results_df.sort_values(by=['R2', 'MAE', 'RMSE'], ascending=[False, True, True])
```

[14]:

	Num_features	R2	MAE	RMSE
172	173	0.828808	17321.418914	30014.293332
173	174	0.828495	17378.620042	30041.684259
171	172	0.828197	17355.091618	30067.771718
170	171	0.828125	17400.417724	30074.046352
169	170	0.828030	17431.113864	30082.414271
174	175	0.827807	17435.749459	30101.923041

168	169	0.827606	17475.317452	30119.417917
176	177	0.827603	17439.740304	30119.729812
167	168	0.827541	17561.356296	30125.116752
175	176	0.827396	17487.048484	30137.762086
166	167	0.826603	17567.306639	30206.958825
165	166	0.826581	17568.329973	30208.858615
163	164	0.826097	17592.331958	30250.985974
164	165	0.826074	17593.919641	30253.015664
179	180	0.824271	17645.938085	30409.418748
178	179	0.824271	17645.962254	30409.419255
180	181	0.824267	17642.573298	30409.764325
177	178	0.824110	17678.855368	30423.296152
160	161	0.821708	17935.006566	30630.328894
161	162	0.821658	17944.372661	30634.636853
181	182	0.821604	17677.958566	30639.249175
162	163	0.821526	17969.664712	30645.959590
153	154	0.821488	17792.117822	30649.257894
152	153	0.821397	17770.857387	30657.058969
159	160	0.821356	17965.000589	30660.603064
154	155	0.821274	17829.870017	30667.602366
155	156	0.821255	17912.562002	30669.198603
158	159	0.821132	17973.246861	30679.762474
157	158	0.821132	17965.703660	30679.818791
156	157	0.820717	17932.969996	30715.350656
182	183	0.820715	17705.086763	30715.555898
184	185	0.819858	17738.611738	30788.825125
183	184	0.819847	17741.564722	30789.816945
144	145	0.818879	17782.897055	30872.416489
142	143	0.818869	17805.514048	30873.209562
141	142	0.818848	17801.188546	30874.999992
146	147	0.818766	17769.000121	30881.999223
143	144	0.818716	17792.728460	30886.247923
147	148	0.818485	17787.824082	30905.959623
150	151	0.818444	17810.934436	30909.454884
149	150	0.818336	17815.483767	30918.635352
148	149	0.818330	17818.647686	30919.109766
145	146	0.818133	17859.068520	30935.942820
151	152	0.818110	17802.604409	30937.835329
185	186	0.818018	17735.116142	30945.701813
186	187	0.818012	17732.465238	30946.181298
188	189	0.817748	17868.154543	30968.610556
187	188	0.817738	17771.386698	30969.452668
136	137	0.817672	17890.053614	30975.078310
129	130	0.817636	17900.596591	30978.118392
134	135	0.817604	17874.448655	30980.832301
135	136	0.817484	17887.407695	30991.048650
199	200	0.817277	18123.433411	31008.653532

198	199	0.817276	18124.770724	31008.695377
133	134	0.817253	17953.729923	31010.624041
139	140	0.817246	17974.622824	31011.291039
140	141	0.817230	17969.040931	31012.600588
137	138	0.817117	18030.692500	31022.173183
200	201	0.817084	18137.609332	31024.958386
131	132	0.816950	17956.365560	31036.360126
130	131	0.816870	17963.256303	31043.175233
138	139	0.816843	18046.206185	31045.434894
201	202	0.816807	18125.897061	31048.481453
197	198	0.816764	18163.989994	31052.162390
132	133	0.816701	17996.467283	31057.476758
196	197	0.816678	18180.454784	31059.396359
210	211	0.816605	18207.330697	31065.588506
208	209	0.816598	18160.662738	31066.173791
211	212	0.816502	18196.733416	31074.314584
189	190	0.816482	17985.578997	31075.990206
209	210	0.816475	18170.462696	31076.602576
206	207	0.816201	18126.875426	31099.822362
204	205	0.816191	18126.626438	31100.612490
202	203	0.816190	18126.521790	31100.757018
203	204	0.816190	18126.521790	31100.757019
205	206	0.816105	18134.197837	31107.874998
207	208	0.815169	18128.054152	31186.984692
191	192	0.815112	18005.040276	31191.807957
190	191	0.815056	18010.127473	31196.537198
192	193	0.814367	18018.942627	31254.609435
193	194	0.813618	18082.995777	31317.578085
194	195	0.813433	18098.768113	31333.078209
195	196	0.812998	18097.366893	31369.580357
124	125	0.812637	18390.334757	31399.892661
128	129	0.811831	18462.718653	31467.316870
127	128	0.811705	18453.012540	31477.848058
125	126	0.811638	18466.605942	31483.506018
126	127	0.811577	18480.548953	31488.577117
103	104	0.809893	19205.704796	31628.967264
212	213	0.809694	18459.118767	31645.505324
104	105	0.809024	19259.372834	31701.196336
121	122	0.808839	19057.945535	31716.563366
101	102	0.808190	19238.321559	31770.279562
122	123	0.807650	19108.487553	31815.038694
123	124	0.807308	19119.429422	31843.270986
102	103	0.806394	19392.614598	31918.726060
80	81	0.805500	19684.863158	31992.359157
79	80	0.805352	19704.063261	32004.486607
120	121	0.804489	19158.714719	32075.361152
117	118	0.804268	19150.500976	32093.508878

112	113	0.804196	19155.070739	32099.409207
111	112	0.804195	19158.501602	32099.427046
113	114	0.804172	19155.764173	32101.358969
119	120	0.804105	19187.803634	32106.805084
118	119	0.804075	19168.788527	32109.290669
115	116	0.804056	19199.984671	32110.834495
114	115	0.804035	19204.994641	32112.593771
78	79	0.803828	19808.141499	32129.523885
110	111	0.803798	19164.945133	32131.970360
99	100	0.803773	19507.750983	32134.071428
116	117	0.803619	19191.968242	32146.623538
109	110	0.803561	19166.588818	32151.382113
98	99	0.803481	19546.382824	32157.921042
77	78	0.803479	19847.049200	32158.137150
100	101	0.803041	19543.245231	32193.906014
106	107	0.802841	19175.192250	32210.300145
107	108	0.802545	19312.239604	32234.451482
105	106	0.802455	19226.908521	32241.806158
97	98	0.802331	19990.759860	32251.930513
108	109	0.802277	19326.399076	32256.277274
81	82	0.802022	20018.416503	32277.122929
74	75	0.802002	19690.126418	32278.721891
82	83	0.801957	20020.733581	32282.363783
73	74	0.801932	19726.545317	32284.457798
72	73	0.801028	19930.252835	32358.021580
94	95	0.800672	20204.044097	32386.930669
95	96	0.800609	20217.512567	32392.031652
96	97	0.800564	20292.308891	32395.699870
75	76	0.800215	20009.752936	32424.034236
83	84	0.800189	19984.627667	32426.185794
71	72	0.799865	20229.468745	32452.473432
64	65	0.799807	20233.545624	32457.128102
76	77	0.799751	20035.285617	32461.726037
84	85	0.799200	19953.411505	32506.358250
93	94	0.799100	20254.917673	32514.431984
70	71	0.799079	20307.099327	32516.125028
68	69	0.798609	20337.052682	32554.107460
69	70	0.798435	20356.834153	32568.182363
92	93	0.798380	20287.579171	32572.594692
90	91	0.798333	20289.748075	32576.430444
91	92	0.798322	20289.430883	32577.339270
85	86	0.798076	20178.052943	32597.197125
86	87	0.798073	20178.482252	32597.390755
89	90	0.797932	20285.729291	32608.797604
65	66	0.797738	20348.584297	32624.477429
87	88	0.797341	20227.740205	32656.430655
67	68	0.797297	20397.968428	32659.980476

88	89	0.797245	20245.231916	32664.145199
66	67	0.797091	20410.839518	32676.540587
31	32	0.795058	20684.842570	32839.908676
63	64	0.794072	20455.847382	32918.740540
62	63	0.793731	20430.903141	32946.000021
227	228	0.793589	18855.970189	32957.394206
214	215	0.793446	18950.846449	32968.807332
215	216	0.793411	18944.409884	32971.595453
213	214	0.793281	18943.238677	32981.934744
30	31	0.792300	20476.026998	33060.127574
219	220	0.792220	18864.655861	33066.465070
218	219	0.791593	18927.480343	33116.343274
217	218	0.790800	18954.204023	33179.266488
225	226	0.790166	18834.049175	33229.533304
226	227	0.790080	18852.070704	33236.285957
216	217	0.789909	18997.799717	33249.816442
39	40	0.789680	20699.633179	33267.932638
42	43	0.789677	20539.783993	33268.216528
223	224	0.789208	18884.266697	33305.279419
222	223	0.789166	18880.950392	33308.584719
224	225	0.789146	18894.133018	33310.173986
40	41	0.788990	20686.821419	33322.461223
41	42	0.788831	20673.036904	33335.063854
58	59	0.788642	20347.850155	33349.984188
221	222	0.788517	18912.730157	33359.829294
57	58	0.788300	20384.494714	33376.925105
220	221	0.788299	18914.561439	33376.987029
47	48	0.788277	20387.977589	33378.752253
46	47	0.788244	20401.288343	33381.315195
61	62	0.788198	20546.200749	33384.932324
229	230	0.788187	18996.306523	33385.823042
230	231	0.788146	19084.769174	33389.065551
228	229	0.788031	18972.164815	33398.101273
26	27	0.787919	20392.336587	33406.927419
60	61	0.787889	20559.052949	33409.309301
27	28	0.787721	20418.369356	33422.552157
45	46	0.787682	20423.211084	33425.636160
59	60	0.787531	20487.905451	33437.461758
44	45	0.787355	20498.945743	33451.371237
24	25	0.787341	20384.436045	33452.475427
23	24	0.787324	20384.979176	33453.742404
25	26	0.787128	20414.833305	33469.194368
22	23	0.787041	20380.707991	33476.067280
36	37	0.786851	21035.867550	33490.988984
38	39	0.786815	21026.536884	33493.795147
29	30	0.786573	20471.197349	33512.825760
43	44	0.786417	20596.820374	33525.028437

37	38	0.786407	21063.480215	33525.828729
28	29	0.786348	20463.115813	33530.468747
32	33	0.785851	21251.520146	33569.421631
33	34	0.785237	21298.941245	33617.506167
21	22	0.784692	20925.976889	33660.184410
49	50	0.784441	20641.387463	33679.796979
50	51	0.784341	20655.196794	33687.562908
48	49	0.784204	20665.163390	33698.262855
51	52	0.783758	20695.900857	33733.094896
55	56	0.783551	20447.255151	33749.209701
53	54	0.782866	20562.384694	33802.567235
20	21	0.782388	21856.041357	33839.736558
35	36	0.782353	21345.631614	33842.496554
34	35	0.781997	21416.790471	33870.177612
54	55	0.781748	20574.056998	33889.495449
56	57	0.781081	20586.913721	33941.258968
52	53	0.779839	20742.336760	34037.401302
231	232	0.774439	19559.527429	34452.286551
19	20	0.772181	22324.901845	34624.264980
18	19	0.770261	22335.782655	34769.907702
17	18	0.769983	22672.432885	34790.940625
16	17	0.769491	22674.295409	34828.077568
243	244	0.766278	16858.305096	35070.004808
244	245	0.766212	16859.475497	35074.965902
15	16	0.764761	23478.326615	35183.653695
241	242	0.763787	18489.345330	35256.353897
240	241	0.762095	18458.609148	35382.467533
239	240	0.761702	18495.820619	35411.683823
238	239	0.757704	18423.130817	35707.436596
236	237	0.757384	18463.141992	35731.022138
4	5	0.757261	23429.850125	35740.097325
10	11	0.756907	23293.253031	35766.124967
242	243	0.756717	18501.722311	35780.114315
237	238	0.756476	18485.204403	35797.863480
5	6	0.756190	23528.747875	35818.860036
14	15	0.754874	23584.664127	35915.392383
9	10	0.754710	23499.054801	35927.423296
13	14	0.754510	23400.710025	35942.014159
12	13	0.754169	23480.429722	35966.998471
11	12	0.754167	23480.280251	35967.115859
7	8	0.752218	23970.890201	36109.414567
6	7	0.752186	24038.424948	36111.778318
8	9	0.752166	23950.010789	36113.231965
235	236	0.737504	19301.287631	37166.122613
232	233	0.737404	19212.011597	37173.218260
233	234	0.737404	19212.011597	37173.218260
234	235	0.737127	19304.370774	37192.774958

3	4	0.730323	25803.376488	37671.094631
2	3	0.722937	26418.122771	38183.429084
1	2	0.693212	27781.861472	40179.523420
0	1	0.644354	32778.190019	43260.864281

Aqui calculo con fuerza bruta con cuantas columnas consigo el mejor coeficiente de determinacion y a su vez un MAE Y RMSE bajos. En mi caso sale que el mejor resultado seria con 174 columnas, asi que en mi caso voy a crear 2 modelos.

1- Muy complejo, pero si se tienen todos los datos entiendo que mas preciso 0.8288

2- Simple y entiendo que lo mas facil pues solo necesita 5 columnas y daria un coeficiente de 0.7572

2. Podemos crear algunas variables para mejorar la relevancia?

En mi caso ya creo que tengo suficientes columnas como para crear mas

1.3.2 Train y test

Dividir los datos en train y test (o train, test y validación) es un paso fundamental para la correcta evaluación de nuestro modelo. Buscamos replicar el efecto de "tener datos nuevos".

Nota: En realidad, lo más correcto seria dividir en train / test ANTES de analizar relevancias - pero simplificamos un poco por ahora.

Nota 2: Si dividimos de forma *random* es importante fijar el seed (te vas a volver loco si no)

1. Qué variables quieres elegir para tu `X` final?

Todas menos SalePrice obviamente

2. Divide los datos en train / test

1.4 Modelado

Ahora vamos a entrenar el modelo. Si has hecho lo anterior bien, este paso es muy sencillo. Normalmente incluimos aquí optimizaciones del modelo, pero para el primer ejercicio esto es de menor importancia.

1.4.1 Entrenar el modelo

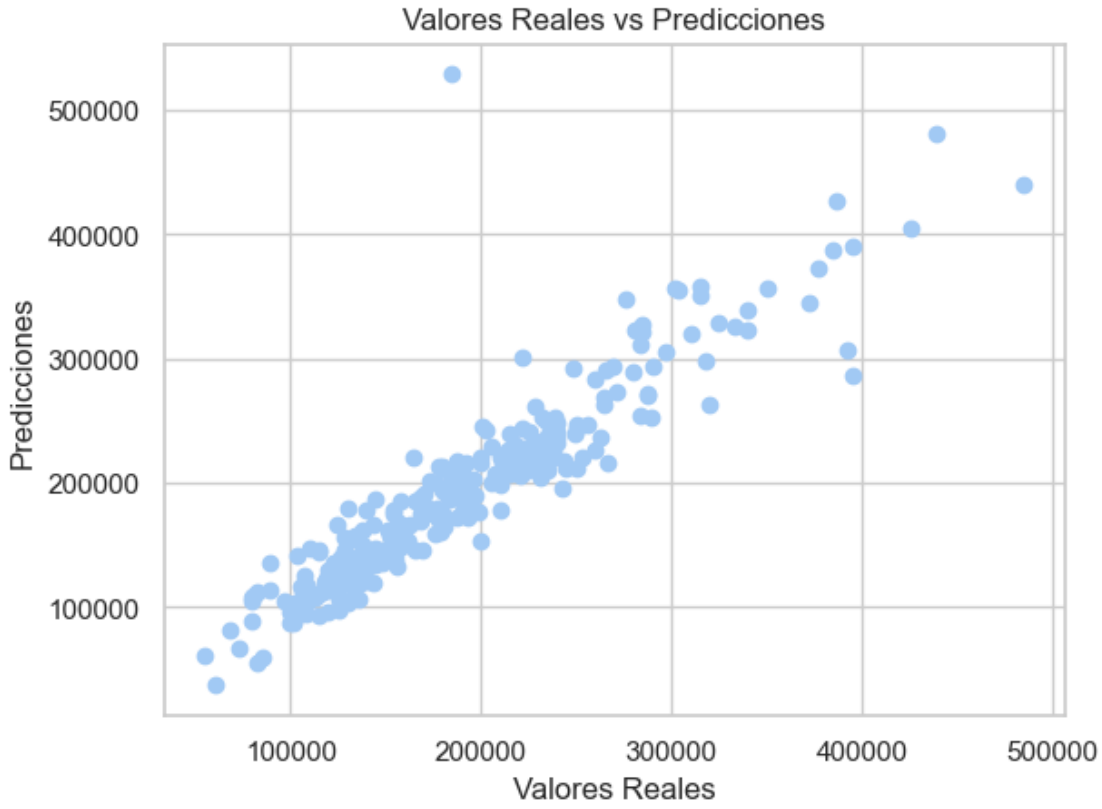
La parta más sexy, pero por ahora lo más sencillo. Tiramos de librerías de modelos ya hechos para simplificarnos la vida.

1. Crea un modelo de regresión

[15]:
```
# df2.drop(523,inplace=True) # Esto mejoraria notablemente el modelo(0.90 de
 ↪coeficiente) pero no quiero borrar una fila que a priori lo unico que tiene
 ↪distinto es un LotArea superior al resto con sus caracteristicas.

X_train, X_test, y_train, y_test = train_test_split(df2[best_features_df.iloc[1:
 ↪174]['Correlation'].tolist()], df2.SalePrice, test_size=0.2, random_state=8)

reg = linear_model.LinearRegression()
```

```
reg.fit(X_train,y_train)

predictions = reg.predict(X_test)

r2 = r2_score(y_test, predictions)
print(f"Coeficiente de Determinación (R^2): {r2}")

plt.scatter(y_test, predictions)
plt.xlabel("Valores Reales")
plt.ylabel("Predicciones")
plt.title("Valores Reales vs Predicciones")
plt.show()
```

```
Coeficiente de Determinación (R^2): 0.8288076349565676
```

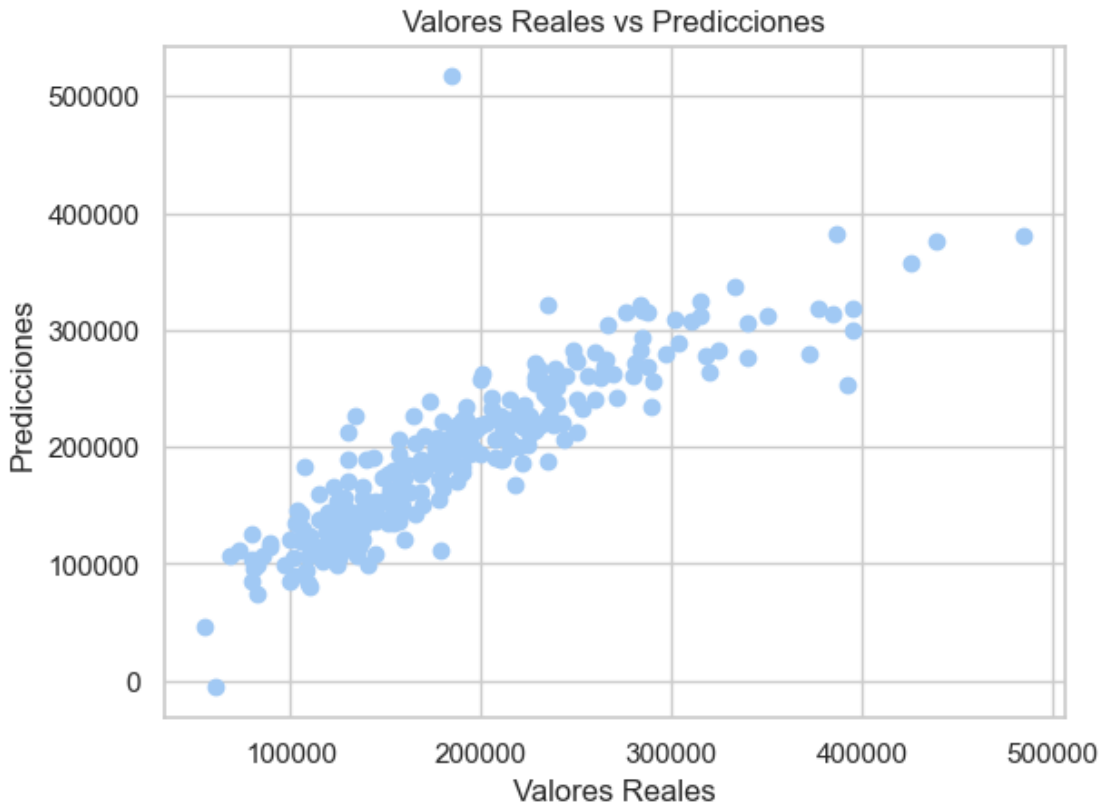


El modelo 1 bastante complejo y que si no se tienen todos los datos totalmente inutil creo

[16]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(df2[best_features_df.
↪iloc[1:6]['Correlation'].tolist()], df2.SalePrice, test_size=0.2,␣
↪random_state=8)
```

```
reg2 = linear_model.LinearRegression()

reg2.fit(X_train2,y_train2)

predictions2 = reg2.predict(X_test2)

r2_2 = r2_score(y_test2, predictions2)
print(f"Coeficiente de Determinación (R^2): {r2_2}")

plt.scatter(y_test2, predictions2)
plt.xlabel("Valores Reales")
plt.ylabel("Predicciones")
plt.title("Valores Reales vs Predicciones")
plt.show()
```

```
Coeficiente de Determinación (R^2): 0.7572609722098338
```



El modelo 2 simple con 5 columnas que a priori son las que mas correlacion tenian con SalePrice

2. Generar predicciones para tu conjunto de test

[17]:
```
data = {
    'OverallQual': [8, 7, 6, 9, 8, 7, 6, 9, 8, 7],
    'GrLivArea': [2000, 1800, 1600, 2200, 2000, 1800, 1600, 2200, 2000, 1800],
    'GarageCars': [2, 2, 2, 3, 2, 2, 2, 3, 2, 2],
    'GarageArea': [500, 480, 460, 600, 500, 480, 460, 600, 500, 480],
    'TotalBsmtSF': [1000, 900, 800, 1100, 1000, 900, 800, 1100, 1000, 900]
}

reg2.predict(pd.DataFrame(data))
```

[17]:
```
array([252443.98913696, 215830.6858017 , 179217.38246645, 306072.81268804,
       252443.98913696, 215830.6858017 , 179217.38246645, 306072.81268804,
       252443.98913696, 215830.6858017 ])
```

1.4.2 Evaluar el modelo

No vamos a dedicar mucho esfuerzo en este paso por ahora, pero es fundamental evaluar las predicciones que hemos generado. El objetivo es entender lo bueno que es nuestro modelo, para entender si podemos fiar de los resultados y usarlo para responder a nuestro problema.

1. Elige las métricas para tu evaluación
2. Evalúa las predicciones generadas

[18]:
```
mae = mean_absolute_error(y_test, predictions)
print(f"Error Absoluto Medio (MAE): {mae}")
```

```
Error Absoluto Medio (MAE): 17321.418913941176
```

[19]:
```
rmse = mean_squared_error(y_test, predictions, squared=False)
print(f"Raíz del Error Cuadrático Medio (RMSE): {rmse}")
```

```
Raíz del Error Cuadrático Medio (RMSE): 30014.29333211952
```

3. El modelo funciona? Se podría utilizar para ayudarnos con el problema que tenemos?

Realmente yo creo que el modelo simple seria el mas util ya que el complejo es absurdamente complejo, aparte creo que cometo el error de sobreentrenarlo, y que tambien hay columnas que existen mas tipos de datos de un tipo que de otro...Tampoco acabo de entender como una sola fila la 523 puede influir tanto en el modelo.