“Master Big Data y Data Science”

Aplicaciones al comercio, empresa y finanzas

Curso 2021 – 2022

Trabajo Fin de Máster:

“House Prices - Advanced Regression Techniques”

Autor: Lourdes Villafaña

Fecha: 22 de septiembre de 2022

GitHub: https://github.com/lourdesv22/UCM-TFM

Google Colab:
https://colab.research.google.com/drive/1sXGwxtI0JNs1dBneeakPgx-m_sCuj9Ae?usp=sharing

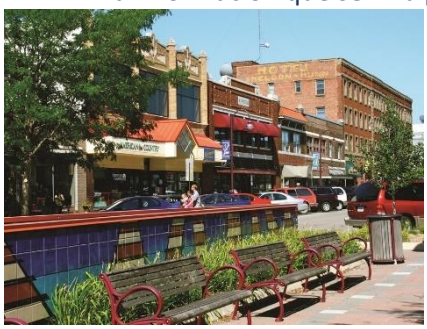
CONTENIDO

1. Comentarios iniciales y objetivos

Este reto es una de las competencias juego de la plataforma Kaggle, que **tiene como objetivo predecir el precio de venta de una propiedad en dólares (SalePrice).** Se evalúa el nivel de ajuste del modelo en base a la raíz del error cuadrático medio (RMSE). El enlace de la competencia es: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview

"Pídale a un comprador de vivienda que describa la casa de sus sueños y probablemente no comenzará con la altura del techo del sótano o la proximidad a una estación de metro que le permita desplazarse. Sin embargo, el conjunto de datos de este análisis demuestra que influye mucho más en las negociaciones el precio, que la cantidad de dormitorios o tener una valla blanca."

La información que servirá para este análisis corresponde a casa residenciales de Ames, Iowa.



Ames es una ciudad situada en el condado de Story, estado de Iowa, Estados Unidos. Según el censo de 2020 tenía una población de 66,427 habitantes.

Los gastos de vivienda de Ames son un 5% más bajos que el promedio nacional y los precios de los servicios públicos son un 8% más bajos que el promedio nacional. Los gastos de transporte, como las tarifas de los autobuses y los precios de la gasolina, son un 6 % más altos que el promedio nacional. Ames tiene precios de comestibles que son 0% más bajos que el promedio nacional. **Ames ha sido incluido en el puesto 15 de los 100 mejores lugares para vivir** de este año, según Livability.com.

Para este análisis, se cuenta con 79 variables explicativas que describen (casi) todos los aspectos de las casas residenciales en Ames, Iowa y **el objetivo es predecir el precio final de cada casa.**

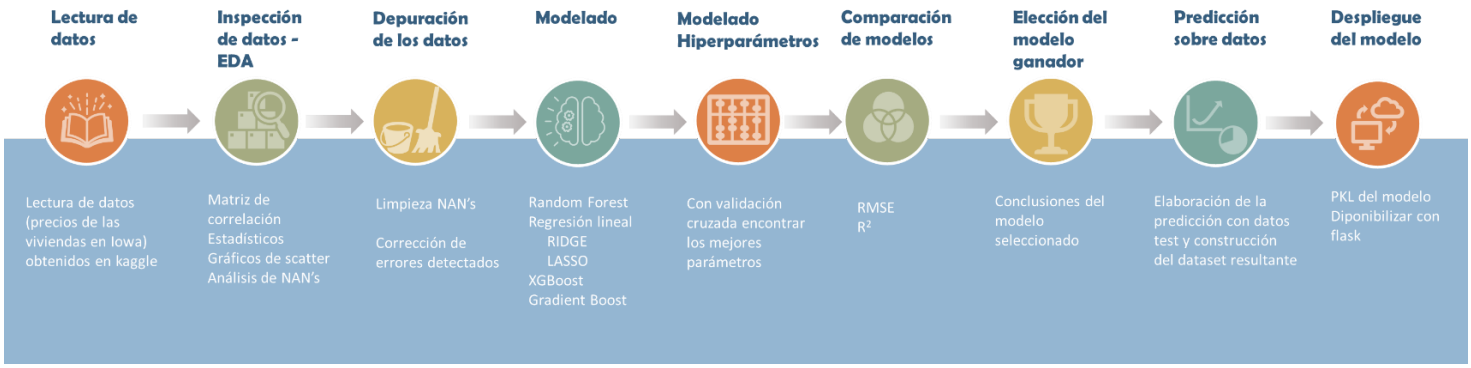
Para lograr nuestro objetivo tenemos:

- Un archivo histórico con información completa para entrenar el modelo ('train.csv')
- Un archivo de casas sin información de precio que servirá para valorar la bondad de nuestro modelo ('test.csv')

El presente trabajo cubre tres funcionalidades:

- Hacer un modelo predictivo basado en datos históricos (training) para predecir los precios de venta de las casas. Por lo tanto, el modelo usará como variable objetivo *'SalePrice'*.
- Crear un API para utilizar las capacidades predictivas del modelo a través de solicitudes HTTP (utilizando Flask, el micro framework de Python para crear aplicaciones web)
- Almacenar la información en una base de datos SQL (Workbench) para tenerla disponible.

Para realizar el estudio se seguirá la siguiente metodología:



2. Lectura de datos

La lectura de los datos, en éste caso fue muy sencilla porque kaggle me da el dataset y descripción de las variables, sin embargo, en la *"vida real"* previo a este punto se debe hacer la recolección de los datos, este paso puede complicarse bastante porque la mayoría de las empresas no cuentan con data organizada/centralizada y dependiendo de la cantidad y calidad es lo bien o mal que puede resultar el modelo.

En México, hay varias regulaciones con el propósito de mantener la confidencialidad y el anonimato de los involucrados. El aviso de privacidad debe hacerse siguiendo lo estipulado en la Ley Federal de Protección de Datos en Posesión de los Particulares (LFPDPPP). Esta hace especial énfasis en los "datos sensibles", tema que deben atender y cuidar puntualmente. Datos personales como el estado civil, información del cónyuge e hijos, religión, preferencia sexual, entre otras piezas de información, no pueden ni deben usarse para ningún fin diferente al que fue expresamente recabado. Las razones son la seguridad de las personas y los derechos humanos para evitar discriminación.

Para el presente trabajo, como ya se mencionó anteriormente, se trabajará con datos que se descargaron del sitio web de kaggle (sitio de la competición).

La lectura de los datos es sencilla y se realizó sin ningún problema, lo que permite avanzar al siguiente paso.

3. Análisis Exploratorio de los Datos (EDA)

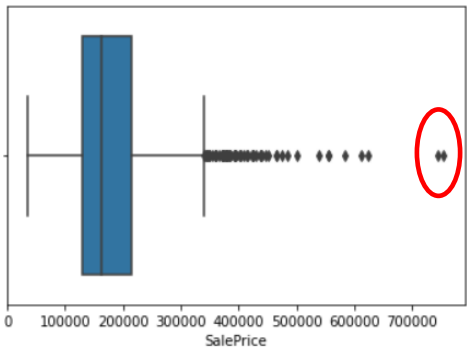
Este estudio está preparado para funcionar con los datos del archivo *Training,* que contiene 81 variables (38 son numéricas y 43 categóricas) que describen varias características para un total de 1,460 casas. La descripción detallada de las variables se puede encontrar en el Anexo I.

La variable objetivo es ***'SalePrice'*** (El precio de venta en dólares), así que voy a analizar sus datos estadísticos:

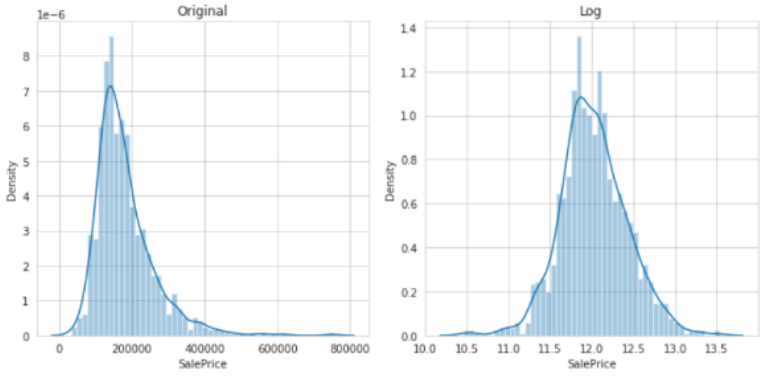
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

Podemos observar que tenemos un conjunto de datos con 1,460 registros donde el **precio promedio** de las casas es de **180,921 usd**, con una **desviación estándar** de **79,442 usd**. El precio mínimo de las casas es de 34,900 usd y el precio máximo es de 755,000 usd.

Observo que la variable objetivo (valores fuera de rango) con



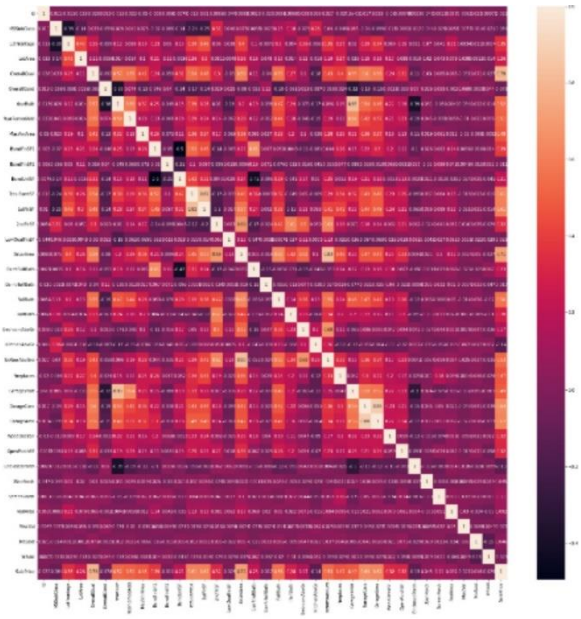
'SalePrice' tiene 2 outliers valor mayor a 700,000.



La **asimetría** es de **1.88** y la **curtosis** es de **6.54**, ambos son positivos, así que podemos asegurar asimetría a la derecha.

Con las gráficas confirmo una distribución sesgada a la derecha, así que le aplico una transformación logarítmica para normalizarla.

La matriz de correlación entre variables nos indica que hay mucha correlación. En el mapa de calor observo que las variables *pies cuadrados totales del sótano* 'TotalBsmtSF' y *pies cuadrados de planta baja* '1stFlrSF' hay una correlación significativa que podría indicar multicolinealidad, es decir, podrían contener la misma información.

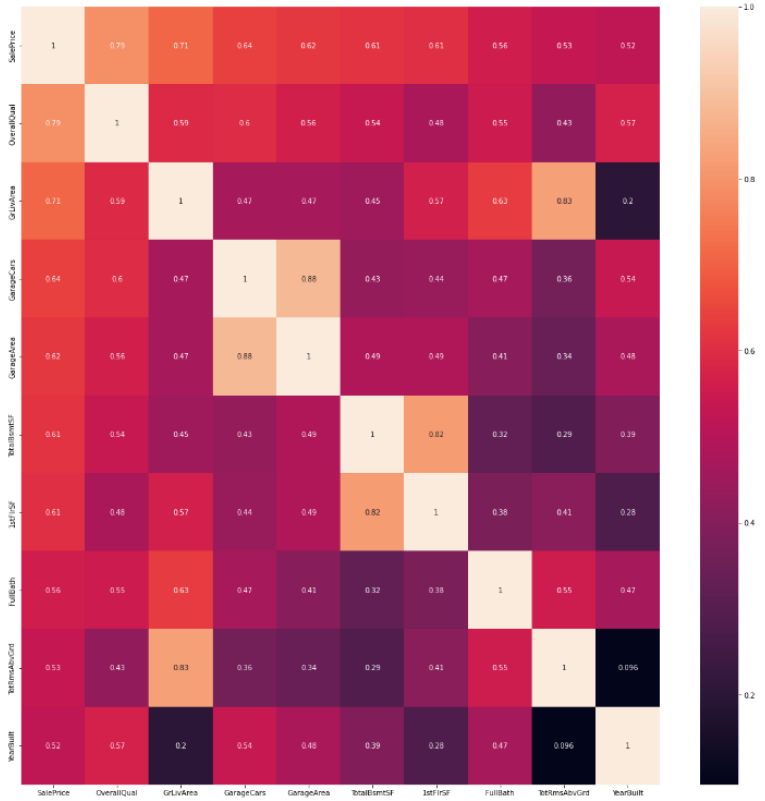


Las correlaciones más grandes están con las siguientes variables:

OverallQual	0.79	Overall material and finish quality
GrLivArea	0.71	Above grade (ground) living area square feet
GarageCars	0.64	Size of garage in car capacity
GarageArea	0.62	Size of garage in square feet
TotalBsmtSF	0.61	Total square feet of basement area
1stFlrSF	0.60	First Floor square feet
FullBath	0.56	Full bathrooms above grade
TotRmsAbvGrd	0.53	Total rooms above grade
YearBuilt	0.52	Original construction date
YearRemodAdd	0.51	Remodel date

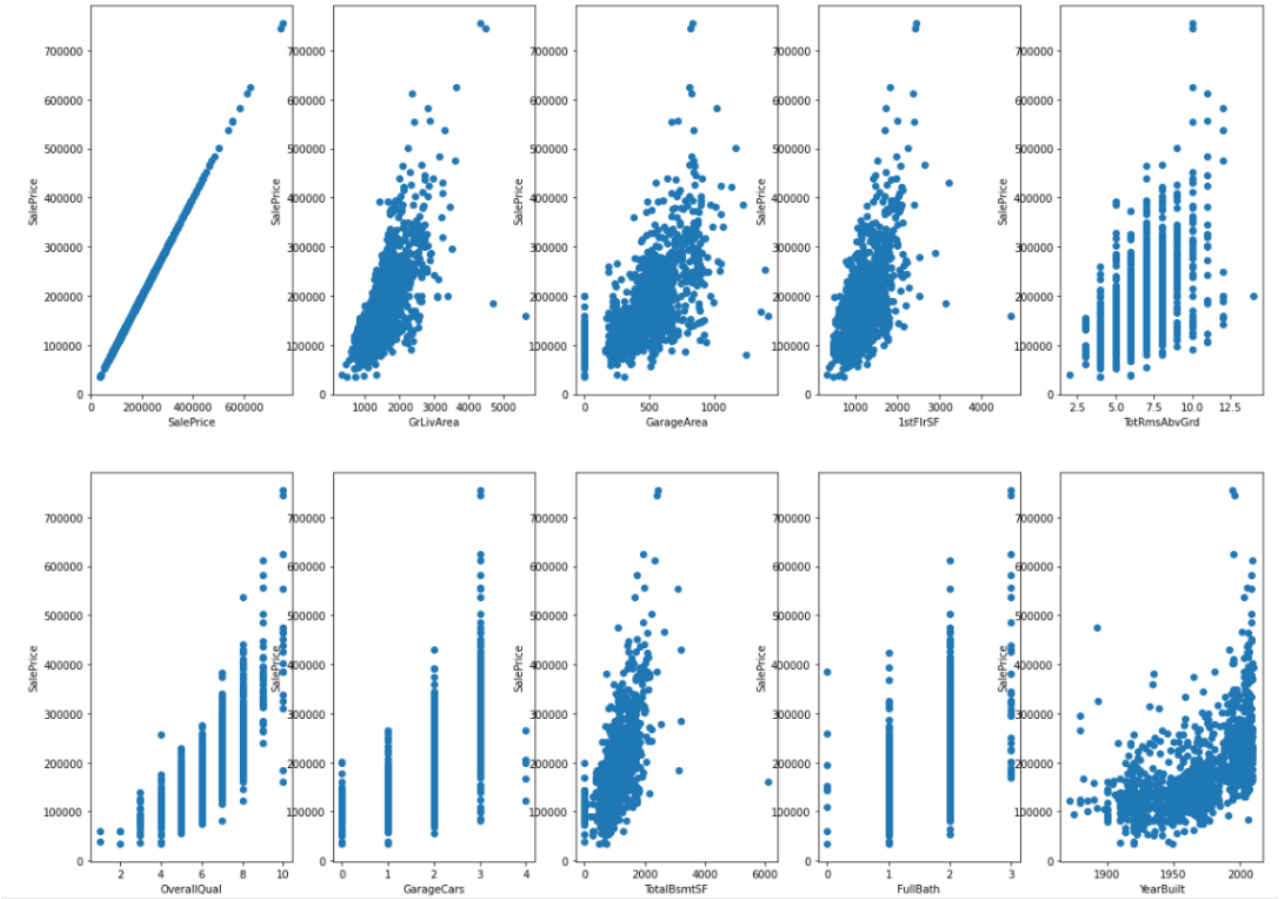
A continuación, me enfoco en las 10 variables más significativas y observo:

El área de la primer planta y la superficie habitable del primer piso tienen una correlación alta de la misma forma esa área total de pies cuadrados de la primer planta con el tamaño del garage y el espacio del sótano, entre otras.

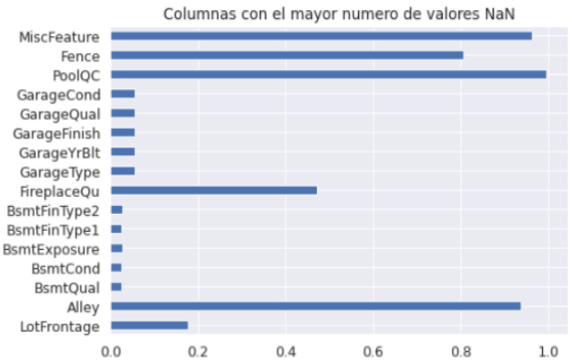


- OverallQual – alta correlación con casi todas las variables
- GrLivArea – alta correlación con TotRmsAbvGrd, FullBath, 1stFlrSF
- GarageCars – alta correlación con GarageYrBlt, YearBuilt and GarageArea.
- GarageArea – alta correlación con GarageYrBuilt, YearBuilt
- TotalBsmtSF – alta correlación con 1stFlrSF
- 1stFlSF – alta correlación con GrLiveArea, GarageArea and TotalBsmtSF
- FullBath – alta correlación con TotalRmsAbvGrade, GrLivArea, GarageYrBlt
- TotRmsAbvGrd – alta correlación con GrLivArea

Además:



- Se puede ver que los datos 'TotRmsAbvGrd', 'OverallQual', 'GarageCars' y 'FullBath' son datos no numéricos
- Los datos de 'GrLivArea', '1stFirSF' y 'TotalBsmtSF' muestran un patrón lineal para 'SalePrice'.

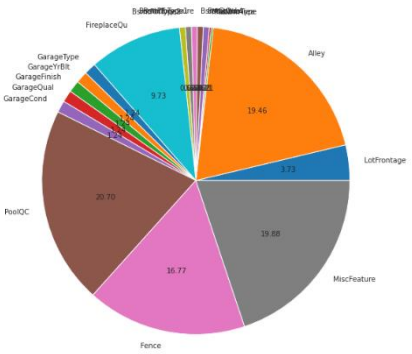


	Total de Nulos	Porcentaje
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
GarageCond	81	0.055479
GarageType	81	0.055479
GarageFinish	81	0.055479

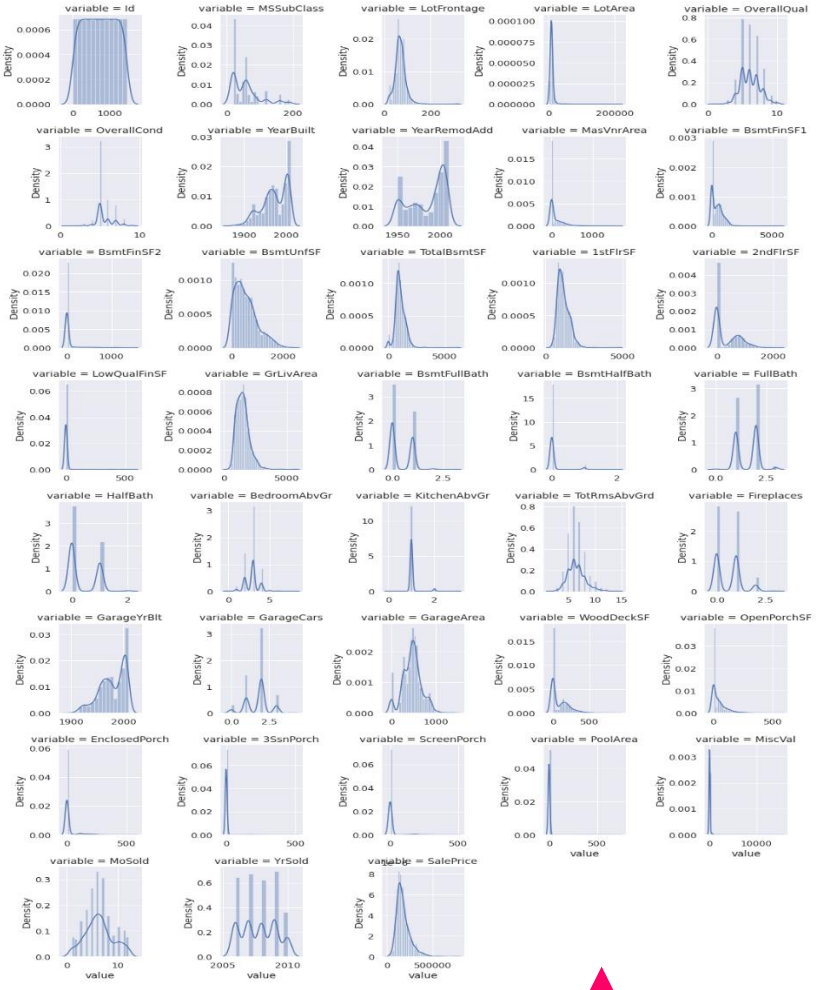
	Total de Nulos	Porcentaje
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtQual	37	0.025342
BsmtCond	37	0.025342
BsmtFinType1	37	0.025342
MasVnrArea	8	0.005479
MasVnrType	8	0.005479
Electrical	1	0.000685
Id	0	0.000000

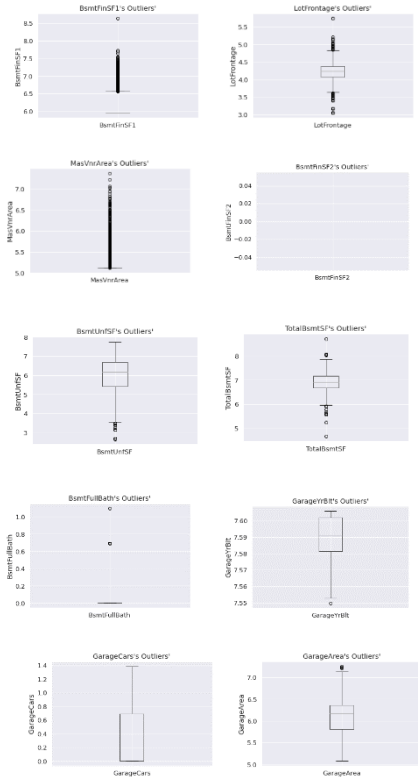
Reviso los registros con NaN (nulos) y veo que las columnas 'PoolQC', 'MiscFeature' 'Alley', 'Fence', 'FireplaceQu' y 'LotFrontage' contienen demasiados valores de este tipo y que pertenecen a la calidad de los materiales de la piscina, valla y otros atributos exteriores de los inmuebles.

Como no conozco detalles sobre la o las causas por las cuales faltan datos, es imposible descartar un posible sesgo y menos estimar su magnitud. Otro punto importante al tratar con datos faltantes, es la cantidad; Si son pocos los datos faltantes, es probable que su efecto sea menor pero si son muchos, su ausencia va comprometiendo progresivamente la validez de las conclusiones. Observé que hay 4 variables que tienen **mas de un 80% de valores NaN**, considero que esas variables será mejor quitarlas.



Como siguiente paso, analizo las **variables numéricas**:

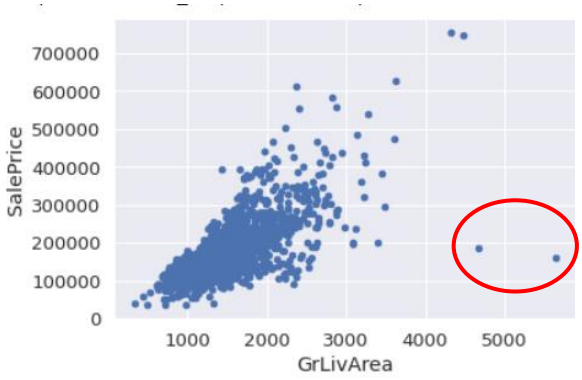




Encuentro que las variables numéricas anteriores tienen muchos valores atípicos, así que decido que completaré los valores faltantes usando su media.

Observo que ninguna de las variables cuantitativas tiene una distribución normal. Algunas variables parecen buenas candidatas para la transformación logarítmica, estas variables son: 'TotalBsmtSF' (Pies cuadrados totales de área de sótano), 'KitchenAbvGr', 'LotFrontage', 'LotArea' y otras.

La transformación suavizará algunas irregularidades que podrían ser importantes, como una gran cantidad de casas con cero *2ndFlrSF*. Sin embargo, las irregularidades son buenas candidatas para la construcción de características.



Al analizar la variable 'GrLivArea' vs 'SalePrice' con un diagrama de dispersión, observo dos valores atípicos, con un precio demasiado bajo en relación con la variable 'GrLivArea'. Debo tener cuidado con esos valores.

Conclusiones

- Se aplicará una transformación logarítmica para normalizar la distribución.
- Las columnas 'PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu' y 'LotFrontage' no tienen información (datos) quizá sea porque las personas no prestan mucha atención a esos aspectos y no se han interesado en recolectar data para esas variables, sin embargo lo importante es que pueden ocasionar que el estudio muestre conclusiones incorrectas, por eso decido que en paso siguiente debo eliminarlas. El criterio usadopara eliminar esta descrito en el paso de data cleaning.
- Las columnas 'GarageCond', 'GarageType', 'GarageYrBlt', 'GarageFinish' y 'GarageQual' tienen el mismo porcentaje de null's.
- Las variables *área de revestimiento con mampostería* 'MasVnrArea' y *tipo de revestimiento de mampostería* 'MasVnrType' tienen correlación positiva con las columnas *calidad general del material* 'OverallQual' y *fecha original de construcción* 'YearBuilt'.
- Los valores atípicos son en relación al precio y la superficie cuadrada habitable del primer nivel, parece que no corresponde un precio tan pequeño a tanta superficie, pero hay que seguir revisando porque hay muchas otras variables.

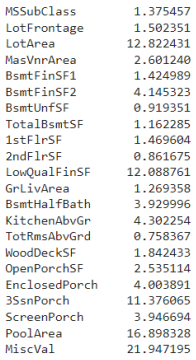
4. Data Preprocessing and Data Cleaning

Continuo con la depuración/corrección de los datos, en base al análisis del punto anterior. Para ello, divido los datos por el tipo de variable: El primer archivo tendrá las variables categóricas y el segundo las variables numéricas.

Al hacer la división de las variables, observo que tengo 37 columnas numéricas y 43 columnas categóricas.

Para las variables numéricas:

1. El primer paso es la reducción de la asimetría, aplicando Log+1a a las variables con sesgo mayor a 0.75. Las columnas modificadas son

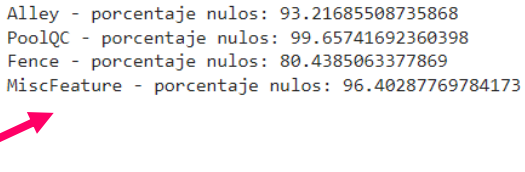


```
MSSubClass       1.375457
LotFrontage      1.502351
LotArea         12.822431
MasVnrArea       2.601240
BsmtFinSF1       1.424989
BsmtFinSF2       4.145323
BsmtUnfSF        0.919351
TotalBsmtSF      1.162285
1stFlrSF         1.469604
2ndFlrSF         0.861675
LowQualFinSF    12.088761
GrLivArea        1.269358
BsmtHalfBath     3.929996
KitchenAbvGr     4.302254
TotRmsAbvGrd     0.758367
WoodDeckSF       1.842433
OpenPorchSF      2.535114
EnclosedPorch    4.003891
3SsnPorch       11.376065
ScreenPorch      3.946694
PoolArea        16.898328
MiscVal         21.947195
```

2. Posteriormente, para el manejo de los valores missing, decido eliminar las columnas que tienen más del 80% de sus valores en missing y llenar con la media cuando son menos del 80%. Después de aplicar la regla y revisar el resultado, observo que no hay variables con más del 80% de nulos así que no se elimina ninguna variable.

Para las variables categóricas:

1. El primer paso es eliminar los valores missing, decido eliminar las columnas que tienen mas de 80% de sus valores en missing y llenar con *'No Aplica'* cuando son menos del 80%. Las variables que se eliminan son:



```
Alley - porcentaje nulos: 93.21685508735868
PoolQC - porcentaje nulos: 99.65741692360398
Fence - porcentaje nulos: 80.4385063377869
MiscFeature - porcentaje nulos: 96.40287769784173
```

2. Posteriormente, utilizo la función función *get_dummies* para eliminar la colinealidad convirtiendo las variables categóricas en variables ficticias o dummies, éste es un paso importante para la transformación y seguir trabajando con los datos antes de empezar con los modelos.

En este momento tengo los dataset transformados (variables numéricas y variables categóricas), como siguiente paso es necesario unirlos para proceder con la selección y entrenamiento del modelo. Al finalizar *tengo 295 variables* (columnas), con las que estaré trabajando.

5. Model Training

Para el modelado, identifiqué que es un **aprendizaje supervisado**, ya que busco reproducir un valor conocido en un conjunto de datos de entrenamiento, así que generaré un **modelo predictivo de regresión** basado en datos de entrada/salida. En este caso utilizaré las ventas históricas para estimar los precios de las casas.

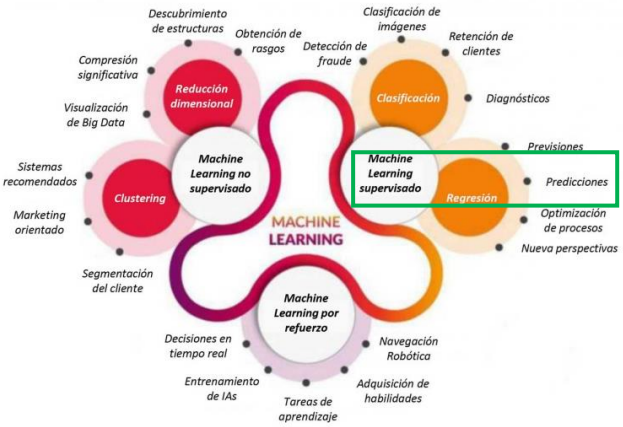
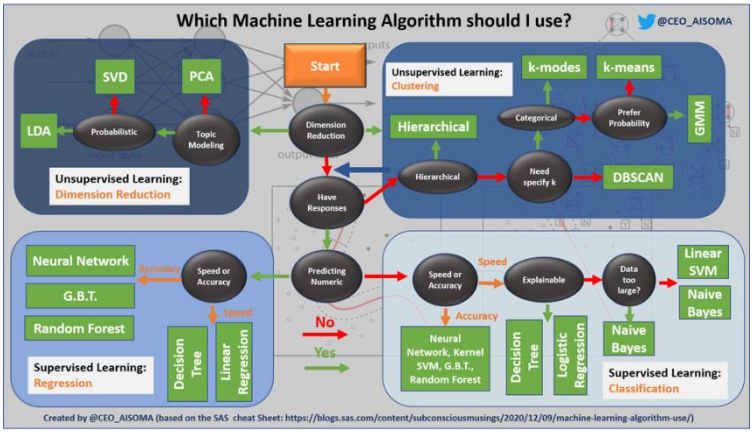
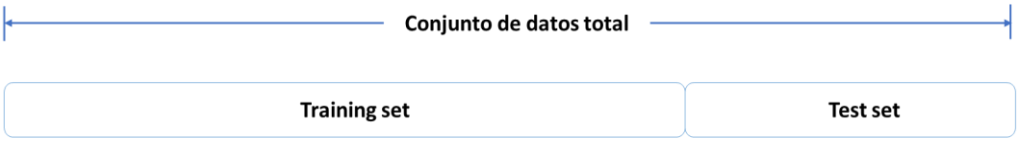


Fig. tomada del curso



Los pasos que seguí para seleccionar el modelo ganador, son los siguientes:

- División de los datos entre datos de **Entrenamiento** (Para seleccionar el modelo, 80%) y datos de **Test** (para probar el modelo final, 20%). Decido un 80-20 para la división del conjunto de datos, quedando la división de las dos partes como se describe a continuación: parte *'train'* de entrenamiento, que corresponde al 80% de nuestro dataset que usaré para entrenar el modelo y una parte *'test'* con 20% para evaluar el modelo entrenado.



- **Elegir los modelos con los que trabajaré y posteriormente seleccionar el mejor:** Para elegir el modelo, tuve en cuenta la precisión, tiempo de entrenamiento y facilidad de uso. **Random Forest, XGBoost y Gradient Boosting** son algoritmos basados en árboles de decisión que no requieren mucho tiempo de entrenamiento y logran una buena precisión superando el problema de ajuste excesivo.

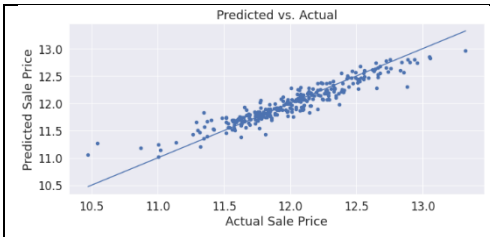
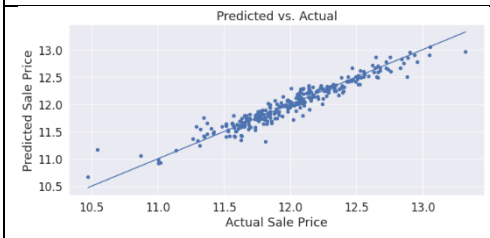
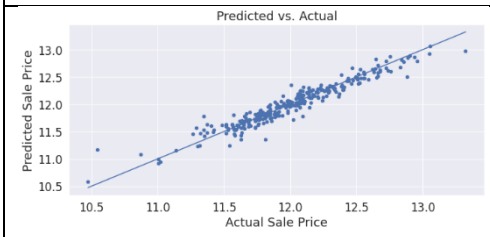
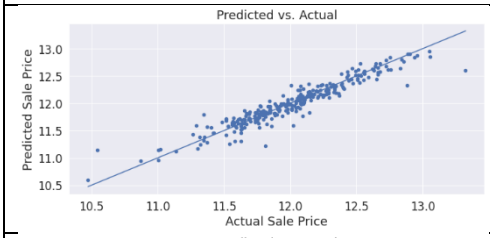
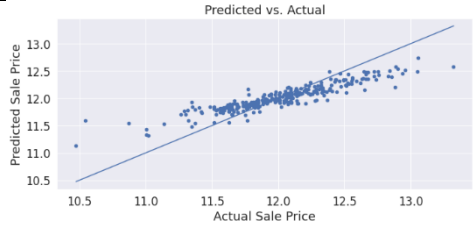
  Adicional, decidí utilizar LASSO y RIDGE porque son modelos de Regresión Lineal que nos ayuda a predecir el futuro en función de la relación pasada de variables. Sin embargo, es importante decir que son muy básicos, por lo tanto, tuve mucho cuidado al analizar su diagrama de dispersión y sus métricas.

  Derivado del análisis anterior, decidí trabajar con los siguientes modelos:
  - Random Forest
  - Regresión Lineal con regularización – RIDGE

- Regresión Lineal con regularización – LASSO
- Xgboost
- GradientBoosting

- Utilizo el set de datos de entrenamiento *'X_train1'* y *'y_train1'* para entrenar el modelo
- **Compruebo la precisión del modelo** con el set de datos de Evaluación que *contiene entradas que el modelo desconoce*. Obtengo métricas del modelo (graficas abajo), en este caso utilice RMSE y R2 principalmente.

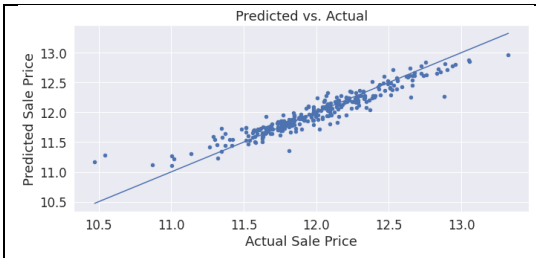
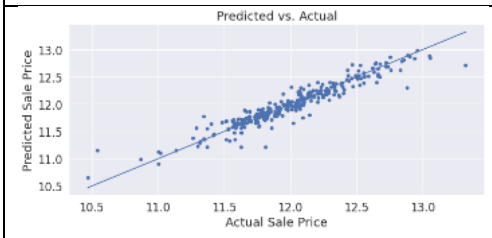
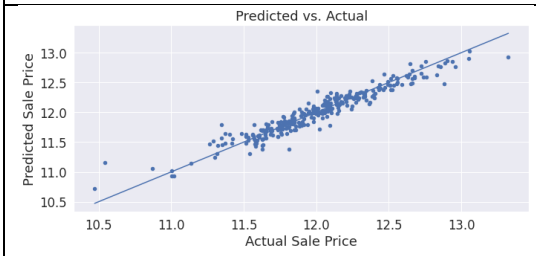
El resumen de las métricas que utilicé para las evaluaciones se presentan a continuación para cada uno de los modelos:

	**Random Forest** RMSE-modelo_RF-CV(7)=0.14572 MSE: 0.021233398421558705 MAE: 0.10425983326461283 R2: 87.94518975842932
	**Regresión Lineal con regularización - RIDGE** RMSE-modelo_ridge-CV(7)=0.11938 MSE: 0.014251421378273085 MAE: 0.08757380298165025 R2: 91.90905869249289
	**Linear regression with regularisation – LASSO** RMSE-modelo_lasso-CV(7)=0.11615 MSE: 0.01348980312441023 MAE: 0.08417541260829861 R2: 92.34145125370962
	**Xgboost** RMSE-modelo_xgb-CV(7)=0.13296 MSE: 0.01767887127545022 MAE: 0.09136342064214284 R2: 89.96319692780189
	**GradientBoosting** RMSE-modelo_gbr-CV(7)=0.22132 MSE: 0.04898404656317153 MAE: 0.1669450676503121 R2: 72.19034963410498

- Utilizo validación cruzada para garantizar que no haya 'overfitting'

Hasta el momento, veo mejores resultados con 'Random Forest', 'XGBoost' y 'Regresión Lineal – LASSO'. A continuación, decido trabajar con la configuración de hiperparámetros para esos 3 modelos.

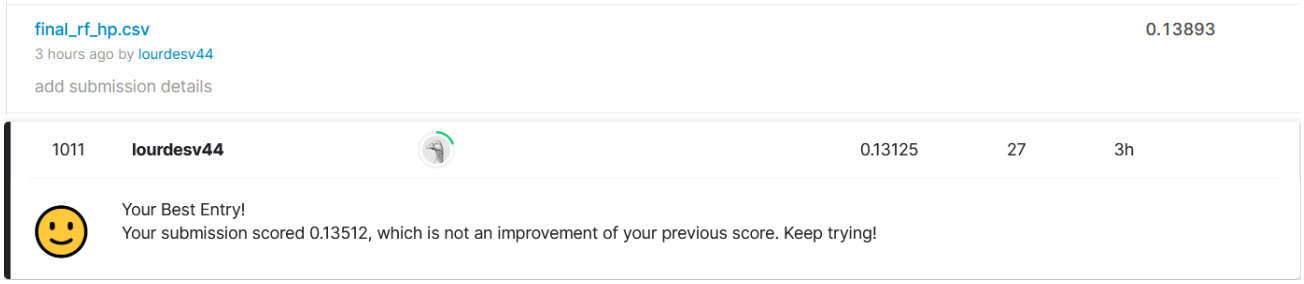
- Configuración de hiperparámetros. Los hiperparámetros de los 3 modelos individuales se optimizaron maximizando la puntuación de validación cruzada utilizando el conjunto de datos de **Entrenamiento**. El resultado fue el siguiente:

	**Random Forest Regressor con hiperparámetros** RMSE-modelo_RF_hp-CV(5)=0.13617+-0.01895 MAE: 0.09567083772785111 R2: 89.21151259582197 **Parámetros:** RandomForestRegressor(max_features=50, n_estimators=500, bootstrap=False)
	**Xgboost con hiperparámetros** RMSE-modelo_xgb_hp-CV(5)=0.13079+-0.01360 MAE: 0.09004644192419467 R2: 90.14607938360028 **Parámetros:** xgb.XGBRegressor(n_estimators=250,learning_rate=0.2,max_depth=2,min_child_weight=1,objective='reg:linear')
	**Linear regression con hiperparametros – LASSO** RMSE-modelo_lasso_hp-CV(5)=0.13170+-0.01738 MAE: 0.08711295333223157 R2: 92.00689463243411 **Parámetros:** linear_model.Lasso(alpha= 0.001, max_iter=100)

- Decido continuar con el modelo de **'Random Forest con hiperparámetros'** así que tomo sus parámetros y entreno el modelo con los datos de **Entrenamiento** y evalúo su rendimiento con los datos de **Test**
- Vuelvo a sacar las métricas a cada modelo para garantizar que mejoraron.
- **El siguiente paso es la predicción:** Utilizo el modelo para obtener la predicción de los valores de nuestros datos objetivo.

Los *scores* obtenidos directamente de la página de Kaggle son los siguientes:

Modelo	LB
Random Forest	0.14458
Regresión Lineal con regularización - RIDGE	5.08556
Linear regression with regularisation – LASSO	3.70863
Xgboost	0.13349
GradientBoosting	0.22494
Decision Tree Regressor con hiperparámetros	0.20724
Random Forest Regressor con hiperparámetros	**0.13893**
Xgboost con hiperparámetros	0.13768



Conclusiones del Modelo

Analicé 5 modelos predictivos de regresión para predecir el precio de las viviendas en Iowa. He mencionado el procedimiento paso a paso para seleccionar el mejor modelo, en todos los casos, entrené el modelo e hice la predicción generando un archivo csv que contenía el precio pronosticado de las viviendas.

Basándome en los modelos que trabaje, comparo los algoritmos por el resultado de raíz cuadrada del error cuadrático medio (RMSE) para seleccionar el que más me convenga. Seleccioné esta métrica porque el RMSE cuantifica cuán diferente es un conjunto de valores (error que hay entre dos conjuntos de datos). Cuanto más pequeño es un valor RMSE, más cercanos son los valores predichos u observados. Sin embargo, evalué y comparé el rendimiento de cada modelo utilizando además las métricas MAE y R2.

En base a lo anterior, decido seleccionar ***'Random Forest con hiperparámetros'*** porque es muy preciso y robusto con el overfitting. Este modelo me da confianza porque considera las variables más importantes y como influyen entre sí.

Para continuar con la siguiente etapa del ejercicio, es necesario guardar el modelo "ganador". Por lo tanto, guardo el modelo en un archivo PKL que me permitirá utilizarlo posteriormente, sin tener que entrenarlo cada vez que se invoque o necesite utilizarlo.

6. Despliegue del modelo

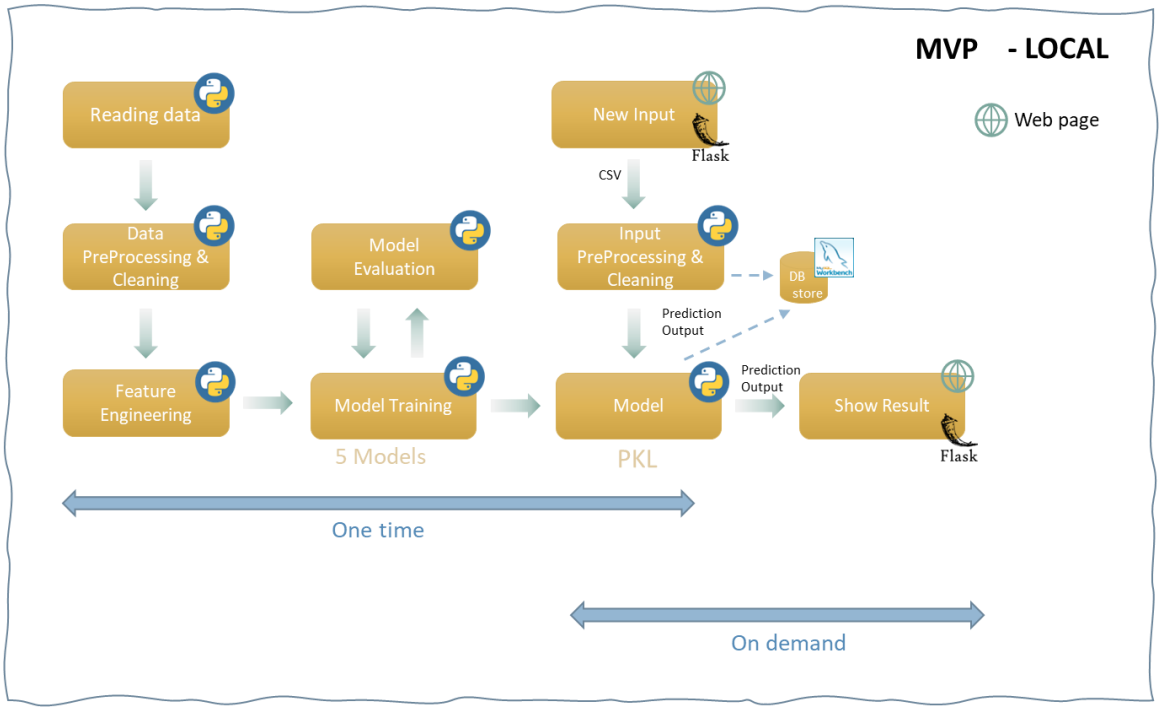
Como parte final del trabajo de fin de master, se busca hacer el despliegue del modelo. Al momento tengo un **modelo entrenado offline**, el siguiente paso es desplegarlo para tenerlo disponible en un servicio donde pueda ser utilizado por el usuario final para predecir en línea.

Después de entrenar el modelo, guardé el modelo como un archivo .pkl para su uso posterior sin tener que entrenarlo nuevamente cada vez que se utiliza. Este proceso llamado *"modelo persistente en formato estándar"* será almacenado en un micro-servicio que se expone para recibir un *request* del cliente.

Para disponibilizar el servicio, desarrollé una página web básica conectada a una base de datos.

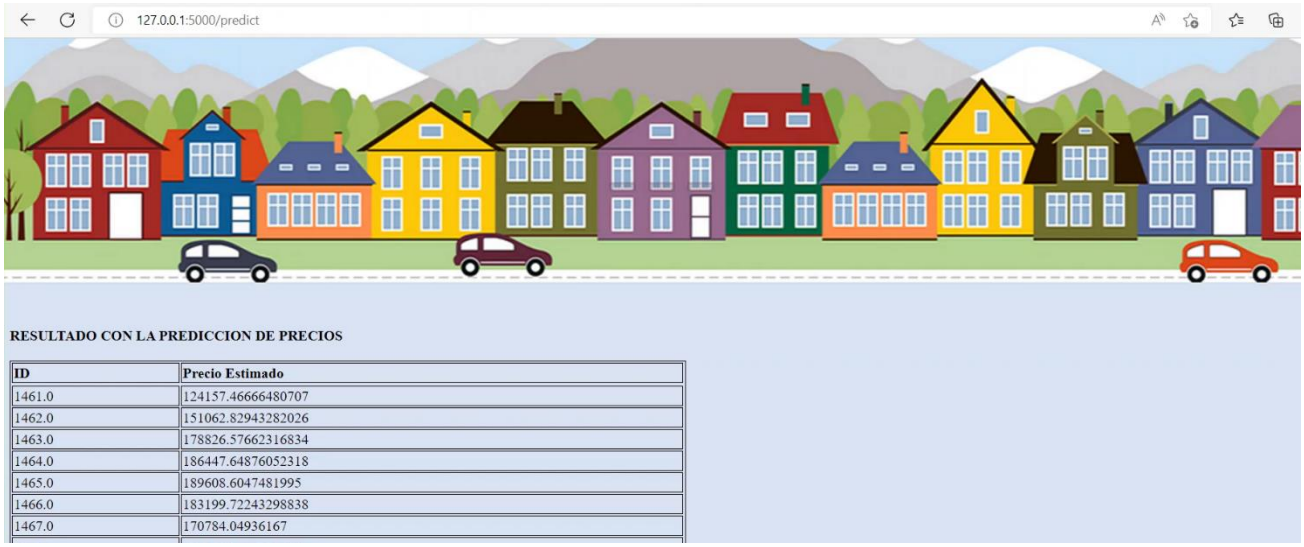
La página me permite cargar un archivo *'csv'* que debe contener las 81 variables para que el modelo pueda dar como resultado la predicción de precio de las viviendas en una segunda página web.

En el siguiente diagrama se muestra el proceso:



Carga de datos para predicción de página web:

Una vez que tengo el archivo PLK con el modelo entrenado, desarrollo una aplicación web que consiste en una página web donde se puede seleccionar el botón de '*Elegir archivo*' para que un usuario pueda subir el archivo en formato 'csv' con los datos de las casas sin precio para que el modelo haga la predicción del costo en dólares. Una vez que se hace el submit del archivo, aparecerá una página web con el ID de las casas y el costo que predijo el modelo.



ID	Precio Estimado
1461.0	124157.46666480707
1462.0	151062.82943282026
1463.0	178826.57662316834
1464.0	186447.64876052318
1465.0	189608.6047481995
1466.0	183199.72243298838
1467.0	170784.04936167

A continuación, se muestra una imagen con la estructura de las carpetas del desarrollo y los componentes:

```
appy_v2.py
funciones.py
templates/
                    home.html
                     result.html
static/
                    /css/styles.css
```

Como se observa en la imagen, existe un **subdirectorio templates** que es donde Flask buscará los archivos HTML de la página de inicio y la página con el resultado.

app_v2.py
Aquí se almacena el código que es ejecutado por el interprete de Python para ejecutar la aplicación web Flask. Aquí se encuentra en código ML para predecir el precio de las casas.

Dentro del código se manda llamar las siguientes dos funciones que están en el archivo:
Funciones_Limpieza.py

- **limpieza:** Esta función hace el pre-processing & data cleaning del archivo csv que introdujo el usuario con los datos de las casas. Adicional inserta en la base de datos la información para que pueda ser consultada posteriormente. Esta función regresa un dataset que sirve de entrada al modelo para predecir el precio de las casas.

- **insertadb**: Esta función actualiza el campo de *'Prediccion'* de la base de datos con el costo que el modelo predijo.

La base de datos está en MySQL Workbench con el nombre 'modelo_casas' con una tabla llamada 'casas' que incluye en total 82 columnas, de las cuales, 81 columnas son del conjunto de datos más 1 columna adicional llamada 'prediccion' para almacenar el resultado del modelo ya que en 'SalePrice' solo se guarda el precio real.

Este desarrollo lo que pretende es ser un MVP (producto mínimo viable) de una aplicación que evolucionaría y llegaría a ser más robusta y que podría ser usada por un agente inmobiliario o un usuario final. De momento y como se explicó **el desarrollo es local**.

En el apartado siguiente se explica brevemente las mejoras y los siguientes desarrollos que se tendrían que hacer a la app.

7. Siguientes pasos y/o posibles mejoras

Modelo:

- Feature Engeeniering: Crear nuevas variables uniendo aquellas que nos permitan mejorar el resultado. Por ejemplo, área total, número total de baños, etc.
- Un *stacking/bagging* de las predicciones por modelos individuales podría mejorar los resultados finales.
- Se puede mejorar el resultado combinando algunos modelos con mejor desempeño pero no correlacionados.

Buscador de precios:

- Aprovechando que la información se guarda en base de datos, lo ideal sería contar con un buscador de precios, así cada vez que el usuario presiona el botón de 'Buscar', el programa 'busca.py' consulta en la base de datos los parámetros de búsqueda y los despliega en la página 'resultados'. Esta opción le permitirá al usuario hacer búsqueda de precios de casas por parámetros *como 'Numero de Baños', 'numero de garanges', 'tamaño'* y obviamente por precio.

Base de datos – Workbench MySQL

- Mejorar el performance
- Crear más tablas y relaciones que soporte el buscador y las demás mejoras

AWS

- Subir el modelo a AWS o una plataforma similar para disponibilizarlo a agentes inmobilarios o usuarios que quieran comprar una casa y quieran consultar precios.
    - Configurar una instancia EC2 en AWS
    - Configurar SQL en AWS y copiar la estructura de la tabla
    - Conectarse a la instancia EC2 de AWS mediante ssh
    - Subir los archivos a la instancia EC2 utilizando copia segura (SCP)
    - Instalar los paquetes necesarios y correr 'app_v2.py' para inicial la app

8. Lecciones aprendidas

1. **Dedicar el tiempo que sea necesario para el análisis de los datos** que nos permitirá tomar las mejores decisiones en el paso de Data Cleaning & Feature Engeeniering. Este paso es clave para el éxito o fracaso de nuestro modelo.
2. Es importante **productivizar los modelos**, para su uso por usuarios finales. Sin olvidar el tema de escalabilidad, disponibilidad y tener un plan de mejora continua.
3. No se necesitan ser una gran empresa de tecnología como Google para recopilar y aprovechar la información; en el master aprendí a sacar provecho de las librerías y paquetes que existen
4. Se debe seguir la metodología para obtener resultados satisfactorios
5. Integración entre todos los pasos

9. Conclusiones

En conclusión, mi mejor puntuación en el sitio de Kaggle fue de 0.13512 y estoy satisfecha porque apliqué los conocimientos adquiridos durante el máster y disfruté mucho siguiendo la metodología que me enseñaron los profesores, con retos y desafíos muy interesantes. Quede sorprendida al encontrar que el *material de construcción y los acabados*, es la variable que más influye al comprar una casa en Iowa. Confirmé lo apasionante que es este mundo de *Big Data & Data Science*, y del que ahora, **gracias a la Universidad y a mis profesores, soy parte activa**.

10. **Referencias**

- Develop a NLP Model in Python & Deploy It with Flask, Step by Step by Susan Li (https://towardsdatascience.com/develop-a-nlp-model-in-python-deploy-it-with-flask-step-by-step-744f3bdd7776)
- Book "Python Machine Learning" by Sebastian Raschka
- stack overflow forum (https://stackoverflow.com/)
- Material compartido por la universidad "Complutense de Madrid" para el Master de Bigdata y Data Science
- SAS (https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/)
- Kaggle competition "House Price – Advanced Regression Techniques" (https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/overview)
- Página web https://www.aprendemachinelearning.com/

Los códigos y datasets utilizados en este trabajo, se encuentran en el siguiente GitHub: https://github.com/lourdesv22/UCM-TFM

Adicional, el código se encuentra disponible en Google Colab: https://colab.research.google.com/drive/1sXGwxtI0JNs1dBneeakPgx-m_sCuj9Ae?usp=sharing

11. Anexo I: Variables

- **SalePrice** – El precio de venta en dólares. Esta es la variable objetivo que se debe predecir.
- **MSSubClass**: clase de construcción
- **MSZoning**: clasificación de la zona
- **LotFrontage**: pies lineales de calle de la parcela
- **LotArea**: tamaño de la parcela en pies cuadrados
- **Street**: tipo de acceso por carretera
- **Alley**: tipo de acceso al callejón
- **LotShape**: forma de la parcela
- **LandContour**: planitud de la parcela
- **Utilities**: servicios públicos disponibles
- **LotConfig**: Configuración de parcela
- **LandSlope**: pendiente de la parcela
- **Neighborhood**: ubicación física dentro de los límites de la ciudad de Ames
- **Condition1**: proximidad a la carretera principal o al ferrocarril
- **Condition2**: proximidad a la carretera principal o al ferrocarril (si hay un segundo)
- **BldgType**: tipo de vivienda
- **HouseStyle**: estilo de vivienda
- **OverallQual**: calidad general del material y del acabado
- **OverallCond**: condición general
- **YearBuilt**: fecha original de construcción
- **YearRemodAdd**: fecha de remodelación
- **RoofStyle**: tipo de cubierta
- **RoofMatl**: material del techo
- **Exterior1st**: revestimiento exterior de la casa
- **Exterior2nd**: revestimiento exterior de la casa (si hay más de un material)
- **MasVnrType**: tipo de revestimiento de mampostería
- **MasVnrArea**: área de revestimiento de mampostería en pies cuadrados
- **ExterQual**: calidad del material exterior
- **ExterCond**: estado del material en el exterior
- **Foundation**: tipo de cimentación
- **BsmtQual**: altura del sótano
- **BsmtCond**: estado general del sótano
- **BsmtExposure**: paredes del sótano a nivel de calle o de jardín
- **BsmtFinType1**: calidad del área acabada del sótano
- **BsmtFinSF1**: pies cuadrados de la superficie acabada tipo 1
- **BsmtFinType2**: calidad de la segunda superficie acabada (si existe)
- **BsmtFinSF2**: Pies cuadrados de la superficie acabada tipo 2
- **BsmtUnfSF**: pies cuadrados del área sin terminar del sótano
- **TotalBsmtSF**: pies cuadrados totales del sótano
- **Heating**: tipo de calefacción
- **HeatingQC**: calidad y estado de la calefacción
- **CentralAir**: aire acondicionado central
- **Electrical**: sistema eléctrico
- **1erFlrSF**: área en pies cuadrados de la primera planta (o planta baja)
- **2ndFlrSF**: área en pies cuadrados de la segunda planta
- **LowQualFinSF**: pies cuadrados acabados de baja calidad (todos los pisos)
- **GrLivArea**: superficie habitable por encima del nivel del suelo en pies cuadrados
- **BsmtFullBath**: cuartos de baño completos en el sótano
- **BsmtHalfBath**: medio baño del sótano
- **FullBath**: baños completos sobre el nivel del suelo

- **HalfBath**: medios baños sobre el nivel del suelo
- **Bedroom**: número de dormitorios por encima del nivel del sótano
- **Kitchen**: número de cocinas
- **KitchenQual**: calidad de la cocina
- **TotRmsAbvGrd**: total de habitaciones por encima del nivel del suelo (no incluye baños)
- **Functional**: valoración de la funcionalidad de la vivienda
- **Fireplaces**: número de chimeneas
- **FireplaceQu**: calidad de la chimenea
- **GarageType**: ubicación del garaje
- **GarageYrBlt**: año de construcción del garaje
- **GarageFinish**: acabado interior del garaje
- **GarageCars**: tamaño del garaje en capacidad de coches
- **GarageArea**: tamaño del garaje en pies cuadrados
- **GarageQual**: calidad de garaje
- **GarageCond**: condición de garaje
- **PavedDrive**: calzada asfaltada
- **WoodDeckSF**: area de plataforma de madera en pies cuadrados
- **OpenPorchSF**: área de porche abierto en pies cuadrados
- **EnclosedPorch**: área de porche cerrada en pies cuadrados
- **3SsnPorch**: área de porche de tres estaciones en pies cuadrados
- **ScreenPorch**: superficie acristalada del porche en pies cuadrados
- **PoolArea**: área de la piscina en pies cuadrados
- **PoolQC**: calidad de la piscina
- **Fence**: calidad de la valla
- **MiscFeature**: característica miscelánea no cubierta en otras categorías
- **MiscVal**: valor en dólares de la característica miscelánea
- **MoSold**: mes de venta
- **YrSold**: año de venta
- **SaleType**: tipo de venta
- **SaleCondition**: Condiciones de venta

12. Anexo II: Código

Librerias

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
from scipy.stats import norm, skew


import sklearn
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.impute import SimpleImputer, MissingIndicator
from sklearn.preprocessing import FunctionTransformer, LabelEncoder, Normalizer, StandardScaler, OneHotEncoder
from sklearn.base import BaseEstimator, TransformerMixin, ClassifierMixin, clone
import sklearn_pandas
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score,train_test_split
from scipy import stats
from sklearn.linear_model import LinearRegression
from scipy.special import boxcox1p
import csv
import seaborn as sns

import sys
import time

import warnings
warnings.filterwarnings('ignore')
```

Importar Datos

```
train_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/TFM/Data/train.csv')
test_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/TFM/Data/test.csv')

# Junto los dos dataframes para hacer el EDA
junto_df = pd.concat((train_df.loc[:,:], test_df.loc[:, :]))
```

```
train_df.head()
```

```
train_df.tail()
```

```
train_df.head()
```

Análisis Exploratorio de los Datos (EDA)

Número de documentos y Columnas

```
print("Tenemos un conjunto de {} documentos (train)".format(len(train_df)))
print("El dataframe tiene {} columnas (train)".format(train_df.shape[1]))
```

```
Tenemos un conjunto de 1460 documentos (train)
El dataframe tiene 81 columnas (train)
```

Información estadística básica de los datos

```
train_df.describe()
```

Información de las columnas

```
train_df.info()
```

```
correlacion = train_df.corr()

plt.subplots(figsize = (25, 25))
sns.heatmap(correlacion, annot = True)
```

```
correlacion_target = correlacion['SalePrice'].sort_values(ascending = False).head(10).index
correlacion_SalePrice = train_df[correlacion_target].corr()

plt.subplots(figsize = (20, 20))
sns.heatmap(correlacion_SalePrice, annot = True)
```

```
fig, axs = plt.subplots(2, 5, figsize = (20, 15))
x, y = 0, 0

for col in correlacion_target:
  axs[x, y].scatter(x = train_df[col], y = train_df['SalePrice'])
  axs[x, y].set_xlabel(col)
  axs[x, y].set_ylabel('SalePrice')

  x += 1

  if x == 2:
    x = 0
    y += 1
```

Análisis de la variable 'SalePrice'

```
train_df['SalePrice'].describe()
```

```
sns.boxplot(train_df['SalePrice'])
```

Análisis de la distribución

```
asimetria = train_df.SalePrice.skew()
kurtosis = train_df.SalePrice.kurt()

print("La asimetria es de {}".format(asimetria))
print("La kurtosis es de {}".format(kurtosis))
```

```
La asimetria es de 1.8828757597682129
La kurtosis es de 6.536281860064529
```

```
%matplotlib inline

sns.set_style("whitegrid")
#sns.set(rc={'axes.facecolor':'white', 'figure.facecolor':'white'})
```

```
#plt.style.use('ggplot')
plt.figure(figsize=(10,5))

plt.subplot(1,2,1)
sns.distplot(train_df.SalePrice, bins=50)
plt.title('Original')

plt.subplot(1,2,2)
sns.distplot(np.log1p(train_df.SalePrice), bins=50)
plt.title('Log')

plt.tight_layout()
```

Análisis de registros con NaN

```
num_nan = train_df.isna().sum() / train_df.shape[0]
```

```
plt.figure(figsize=(8, 5))
sns.set(font_scale=1.2)
num_nan[num_nan > 0.01].plot(kind = "barh")
plt.title("Columnas con el mayor numero de valores NaN")
```

Análisis de la variable 'GrLivArea'

```
analisis_GrLivArea = pd.concat([train_df["SalePrice"], train_df['GrLivArea']], axis=1)
analisis_GrLivArea.head()
```

```
analisis_GrLivArea.plot.scatter(x='GrLivArea', y='SalePrice')
```

Análisis de columnas con valores NULL

```
total_null = train_df.isnull().sum().sort_values(ascending = False)
porcentaje_null = (total_null / train_df.isnull().count()).sort_values(ascending = False)

valores_null = pd.concat([total_null, porcentaje_null], axis = 1, keys = ['Total de Nulos', 'Porcentaje'])
valores_null.head(20)
```

```
train_df.shape
```

```
train_df_dummies = pd.get_dummies(train_df)
train_df_dummies['SalePrice']
```

Data Preprocessing and Data Cleaning

```
train_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/TFM/Data/train.csv')
test_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/TFM/Data/test.csv')

# Junto los dos dataframes para hacer el EDA
junto_df = pd.concat((train_df.loc[:,:], test_df.loc[:, :]))
```

```
sns.set_style("whitegrid")
```

```
target = train_df['SalePrice']
target_log = np.log1p(train_df['SalePrice'])
```

Para aplicar las transformaciones, utilizaré el dataset donde tengo junto train y test.

```
train_df = train_df.drop(["SalePrice"], axis=1)
junto_df = pd.concat([train_df, test_df], ignore_index=True)
```

Divido el dataset en 2: El primero tendrá las variables categóricas y el segundo las variables numericas

```
categoricas = [col for col in junto_df.columns.values if junto_df[col].dtype == 'object']

categoricas_df = junto_df[categoricas]
numericas_df = junto_df.drop(categoricas, axis=1)
```

```
numericas_df.head(1)
```

```
numericas_df.describe()
```

```
categoricas_df.head(1)
```

Reducción de Asimetrías para variables numéricas

```
num_skew = numericas_df.apply(lambda x: skew(x.dropna()))
num_skew = num_skew[num_skew > 0.75]

#Aplico Log+1a a las variables con sesgo > 0.75
numericas_df[num_skew.index] = np.log1p(numericas_df[num_skew.index])
```

```
num_skew
```

Manejo de valores Missing: Variables Numericas

```
data_len = numericas_df.shape[0]

for col in numericas_df.columns.values:
    missing_values = numericas_df[col].isnull().sum()

    # drop si tiene mas de 50 valores valores missing
    if missing_values > 50:
        numericas_df = numericas_df.drop(col, axis = 1)
    # Si es menor a 50, lo lleno con la media
    else:
        numericas_df = numericas_df.fillna(numericas_df[col].median())
```

```
numericas_df.describe()
```

Manejo de valores Missing: Variables Categóricas

```
data_len = categoricas_df.shape[0]

for col in categoricas_df.columns.values:
    missing_values = categoricas_df[col].isnull().sum()

    # drop si tiene mas de 50 valores missing
    if missing_values > 50:
        print("Eliminando columna: {}".format(col))
        categoricas_df = categoricas_df.drop(col, axis = 1)

    # Si es menor a 50, lo lleno con 'No Aplica'
    else:
        categoricas_df = categoricas_df.fillna('No aplica')
        pass
```

```
categoricas_df.describe()
```

```
categoricas_df_dummies = pd.get_dummies(categoricas_df)
```

```
categoricas_df.shape
```

```
datos = pd.concat([numericas_df, categoricas_df_dummies], axis=1)
#datos = pd.concat([numericas_df, categoricas_df], axis=1)
```

Model Training

Divido el dataset en train y test, seleccionando 80 - 20

```
X_train = train_bis[train_bis.columns.values[1:-1]]
y_train = train_bis[train_bis.columns.values[-1]]

# Quito ID
X_test = test_bis[test_bis.columns.values[1:]]
```

```
# Selecciono 80-20
X_train1, X_test1, y_train1, y_test1 = train_test_split(X_train, y_train, test_size = 0.2, random_state = 10)
```

El primer modelo lo haré con Random Forest

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

%matplotlib inline

modelo_RF = RandomForestRegressor(n_estimators=500, n_jobs=-1)

modelo_RF.fit(X_train1, y_train1)
rf_pred = modelo_RF.predict(X_test1)

#Gráfica
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, rf_pred, s=20)
```

```
plt.title('Predicted vs. Actual')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import math
MSE_RF = mean_squared_error(y_test1, rf_pred)
RMSE_RF = math.sqrt(MSE_RF)

print("RMSE-{}-CV({})={:06.5f}".format('modelo_RF', '7', RMSE_RF))
print('MSE:', MSE_RF)
print('MAE:', mean_absolute_error(y_test1, rf_pred))
print('R2: ',  r2_score(y_test1, rf_pred) * 100)

#Estadistica
print('Random Forest Regressor')
print('RMSE :', np.sqrt(mean_squared_error(y_test1, rf_pred)))
print('MAE :', mean_absolute_error(y_test1, rf_pred))
print('R2 :', r2_score(y_test1, rf_pred) * 100)
```

```
Random Forest Regressor
RMSE-modelo_RF-CV(7)=0.14572
MSE: 0.021233398421558705
MAE: 0.10425983326461283
R2:  87.94518975842932
```

```
#Aplico Cross-Validation para validar overfiting
RF_CV = cross_val_score(modelo_RF, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(RF_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_RF', '7', RF_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_RF', '7', RF_CV.std()))
```

```
rf_pred_log = modelo_RF.predict(X_test)

#Genero el archivo test con la predicción
final_rf = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(rf_pred_log)})
final_rf.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_rf.csv", index=False)
```

El segundo modelo es Regresión Lineal con regularización - RIDGE

```
from sklearn.linear_model import Ridge, RidgeCV
modelo_ridge = Ridge()
alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]

def calcula_alpha(model):
    res_alpha = np.sqrt(-cross_val_score(model, X_train, y_train, scoring="neg_mean_squared_error", cv = 7))
    return(res_alpha)

cv_ridge = [calcula_alpha(Ridge(alpha = alpha)).mean() for alpha in alphas]
```

```
cv_ridge
```

```
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validation")
```

```
sns.set(font_scale=1.5)
plt.xlabel("alpha")
plt.ylabel("rmse")
```

```
cv_ridge
```

```
y_test_pred_log = modelo_ridge.predict(X_test)
modelo_ridge = Ridge(alpha = 10).fit(X_train1, y_train1)
ridge_pred = modelo_ridge.predict(X_test1)

#Gráfica
sns.set(font_scale=1.5)
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, ridge_pred, s=20)
plt.title('Predicted vs. Actual')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import math
MSE_ridge = mean_squared_error(y_test1, ridge_pred)
RMSE_ridge = math.sqrt(MSE_ridge)

print("RMSE-{}-CV({})={:06.5f}".format('modelo_ridge', '7', RMSE_ridge))
print('MSE:', MSE_ridge)
print('MAE:', mean_absolute_error(y_test1, ridge_pred))
print('R2: ',  r2_score(y_test1, ridge_pred) * 100)
```

```
#Aplico Cross-Validation para validar overfiting
ridge_CV = cross_val_score(modelo_ridge, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(ridge_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_ridge', '7', ridge_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_ridge', '7', ridge_CV.std()))
```

```
ridge_pred_log = modelo_ridge.predict(X_test)

#Genero el archivo test con la predicción
final_ridge = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(ridge_pred_log)})
final_ridge.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_ridge.csv", index=False)
```

El tercer modelo es Xgboost LB: 0.13429

```
from sklearn.linear_model import LassoCV
modelo_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train1, y_train1)

modelo_lasso.fit(X_train1, y_train1)
lasso_pred = modelo_lasso.predict(X_test1)

#Gráfica
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, lasso_pred, s=20)
plt.title('Predicted vs. Actual')
```

```
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
MSE_lasso = mean_squared_error(y_test1, lasso_pred)
RMSE_lasso = math.sqrt(MSE_lasso)

print("RMSE-{}-CV({})={:06.5f}".format('modelo_lasso', '7', RMSE_lasso))
print('MSE:', MSE_lasso)
print('MAE:', mean_absolute_error(y_test1, lasso_pred))
print('R2: ',  r2_score(y_test1, lasso_pred) * 100)
```

```
#Aplico Cross-Validation para validar overfiting
lasso_CV = cross_val_score(modelo_lasso, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(lasso_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_lasso', '7', lasso_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_lasso', '7', lasso_CV.std()))
```

```
lasso_pred_log = modelo_lasso.predict(X_test)
final_lasso = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(lasso_pred_log)})
final_lasso.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_lasso.csv", index=False)
final_lasso.head()
```

El cuarto modelo es GradientBoosting (LB: 0.21638)

```
import xgboost as xgb

dtrain = xgb.DMatrix(X_train1, label = y_train1)
dtest = xgb.DMatrix(X_test1)

params = {"max_depth":2, "eta":0.1}
modelo_xgb = xgb.cv(params, dtrain,  num_boost_round=500, early_stopping_rounds=100)
```

```
modelo_xgb.loc[30:,["test-rmse-mean", "train-rmse-mean"]].plot()
```

```
modelo_xgb = xgb.XGBRegressor(n_estimators=360, max_depth=2, learning_rate=0.1)

modelo_xgb.fit(X_train1, y_train1)
xgb_pred = modelo_xgb.predict(X_test1)

#Gráfica
sns.set(font_scale=1.5)
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, xgb_pred, s=20)
plt.title('Predicted vs. Actual')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
MSE_xgb = mean_squared_error(y_test1, xgb_pred)
RMSE_xgb = math.sqrt(MSE_xgb)
```

```
print("RMSE-{}-CV({})={:06.5f}".format('modelo_xgb', '7', RMSE_xgb))
print('MSE:', MSE_xgb)
print('MAE:', mean_absolute_error(y_test1, xgb_pred))
print('R2: ',  r2_score(y_test1, xgb_pred) * 100)
```

```
#Aplico Cross-Validation para validar overfiting
xgb_CV = cross_val_score(modelo_xgb, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(xgb_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_xgb', '7', xgb_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_xgb', '7', xgb_CV.std()))
```

```
xgb_pred_log = modelo_xgb.predict(X_test)
final_xgboost = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(xgb_pred_log)})
final_xgboost.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_xgboost.csv", index=False)
final_xgboost.head()
```

El quinto modelo es GradientBoosting

```
from sklearn.ensemble import GradientBoostingRegressor
parametros = {'n_estimators': 100, 'max_depth': 10, 'min_samples_split': 2,
          'learning_rate': 0.01, 'loss': 'ls'}

modelo_gbr = GradientBoostingRegressor(**parametros)
modelo_gbr.fit(X_train1, y_train1)
gbr_pred = modelo_gbr.predict(X_test1)

#Gráfica
sns.set(font_scale=1.5)
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, gbr_pred, s=20)
plt.title('Predicted vs. Actual')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
MSE_gbr = mean_squared_error(y_test1, gbr_pred)
RMSE_gbr = math.sqrt(MSE_gbr)

print("RMSE-{}-CV({})={:06.5f}".format('modelo_gbr', '7', RMSE_gbr))
print('MSE:', MSE_gbr)
print('MAE:', mean_absolute_error(y_test1, gbr_pred))
print('R2: ',  r2_score(y_test1, gbr_pred) * 100)
```

```
#Aplico Cross-Validation para validar overfiting
gbr_CV = cross_val_score(modelo_gbr, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(gbr_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_gbr', '7', gbr_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_gbr', '7', gbr_CV.std()))
```

```
gbr_pred_log = modelo_gbr.predict(X_test)
final_gbr = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(gbr_pred_log)})
final_gbr.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_gbr.csv", index=False)
```

```
final_gbr.head()
```

Hiperparámetros

A continuación, los hiperparámetros de los modelos individuales los optimizaré con validación cruzada.

```
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import make_scorer
```

```
def rmse_cv(modelo, X_train1, y_train1):
    kfold = KFold(n_splits=folds, random_state=seed, shuffle=True)
    rmse = np.sqrt(-cross_val_score(modelo, X_train1, y_train1, scoring="neg_mean_squared_error", cv = kfold))
    return(rmse)

def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

folds = 5
seed = 7
```

Decision Tree Regressor con hiperparámetros

```
from sklearn.tree import DecisionTreeRegressor

# tunning
modelo_DTR_hp = DecisionTreeRegressor()

param_grid = {
        'max_depth': range(2, 11, 2),
        'min_samples_split': [2, 3, 4],
        'min_samples_leaf': range(1, 20, 5),
        'max_leaf_nodes': range(2, 20, 5)
}

kfold = KFold(n_splits=folds, random_state=seed, shuffle=True)

scorer = make_scorer(rmse, greater_is_better=False)
grid_search = GridSearchCV(modelo_DTR_hp, param_grid, n_jobs=-1, cv=kfold, verbose=1, scoring=scorer)
grid_result = grid_search.fit(X_train1, y_train1)

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
  print("{:06.5f} ({:06.5f}) with {}".format(mean, stdev, param))

# Resultados
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# Modelo
modelo_DTR_hp = DecisionTreeRegressor(max_depth=6, min_samples_split=2, max_leaf_nodes=17, min_samples_leaf=16
)

# Fit & Predict
modelo_DTR_hp.fit(X_train1, y_train1)
```

```
DTR_hp_pred = modelo_DTR_hp.predict(X_test1)

#Gráfica
sns.set(font_scale=1.5)
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, DTR_hp_pred, s=20)
plt.title('Predicted vs. Actual')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
RMSE_DTR_hp = rmse_cv(modelo_DTR_hp, X_train1, y_train1)

print("RMSE-{}-CV({})={:06.5f}+-
{:06.5f}".format('modelo_DTR_hp', folds, RMSE_DTR_hp.mean(), RMSE_DTR_hp.std()))
print('MAE:', mean_absolute_error(y_test1, DTR_hp_pred))
print('R2: ',  r2_score(y_test1, DTR_hp_pred) * 100)
```

```
#Aplico Cross-Validation para validar overfiting
DTR_hp_CV = cross_val_score(modelo_DTR_hp, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(DTR_hp_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_DTR_hp', '7', DTR_hp_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_DTR_hp', '7', DTR_hp_CV.std()))
```

```
# Predicción
DTR_hp_pred_log = modelo_DTR_hp.predict(X_test)
final_DTR_hp = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(DTR_hp_pred_log)})
final_DTR_hp.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_dtr_hp.csv", index=False)
final_DTR_hp.head()
```

Random Forest Regressor con hiperparámetros

```
# tunning
modelo_RF_hp = RandomForestRegressor()

param_grid = {
    'bootstrap': [True, False],
    'n_estimators': [100, 500, 1000],
    'max_features': [5, 50]
}

kfold = KFold(n_splits=folds, random_state=seed, shuffle=True)

scorer = make_scorer(rmse, greater_is_better=False)
grid_search = GridSearchCV(modelo_RF_hp, param_grid, n_jobs=-1, cv=kfold, verbose=1, scoring=scorer)
grid_result = grid_search.fit(X_train1, y_train1)

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("{:06.5f} ({:06.5f}) with {}".format(mean, stdev, param))

# Resultados
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# Modelo
modelo_RF_hp = RandomForestRegressor(max_features=50, n_estimators=500, bootstrap=False)

# Fit & Predict
modelo_RF_hp.fit(X_train1, y_train1)
RF_hp_pred = modelo_RF_hp.predict(X_test1)

#Gráfica
sns.set(font_scale=1.5)
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, RF_hp_pred, s=20)
plt.title('Predicted vs. Actual')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
RMSE_RF_hp = rmse_cv(modelo_RF_hp, X_train1, y_train1)

print("RMSE-{}-CV({})={:06.5f}+-{:06.5f}".format('modelo_RF_hp', folds, RMSE_RF_hp.mean(), RMSE_RF_hp.std()))
print('MAE:', mean_absolute_error(y_test1, RF_hp_pred))
print('R2: ',  r2_score(y_test1, RF_hp_pred) * 100)
```

```
#Aplico Cross-Validation para validar overfiting
RF_hp_CV = cross_val_score(modelo_RF_hp, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(RF_hp_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_DTR_hp', '7', RF_hp_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_DTR_hp', '7', RF_hp_CV.std()))
```

```
# Predicción
RF_hp_pred_log = modelo_RF_hp.predict(X_test)
final_RF_hp = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(RF_hp_pred_log)})
final_RF_hp.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_RF_hp.csv", index=False)
final_RF_hp.head()
```

Xgboost con hiperparámetros (LB: 0.13495)

```
# tunning
modelo_xgb_hp = xgb.XGBRegressor()

param_grid = {
    'max_depth': [2, 4],
    'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
    'min_child_weight': range(1, 10, 2),
    'n_estimators': range(50, 300, 50),
    'objective': ['reg:linear']
}

kfold = KFold(n_splits=folds, random_state=seed, shuffle=True)

scorer = make_scorer(rmse, greater_is_better=False)
grid_search = GridSearchCV(modelo_xgb_hp, param_grid, n_jobs=-1, cv=kfold, verbose=1, scoring=scorer)
grid_result = grid_search.fit(X_train1, y_train1)

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
```

```
for mean, stdev, param in zip(means, stds, params):
    print("{:06.5f} ({:06.5f}) with {}".format(mean, stdev, param))

# Resultados
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# Model
modelo_xgb_hp = xgb.XGBRegressor(n_estimators=250,learning_rate=0.2,max_depth=2,min_child_weight=1,objective='
reg:linear')

# Fit & Predict
modelo_xgb_hp.fit(X_train1, y_train1, verbose=True)
xgb_hp_pred = modelo_xgb_hp.predict(X_test1)

#Gráfica
sns.set(font_scale=1.5)
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, xgb_hp_pred, s=20)
plt.title('Predicted vs. Actual')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
RMSE_xgb_hp = rmse_cv(modelo_xgb_hp, X_train1, y_train1)

print("RMSE-{}-CV({})={:06.5f}+-
{:06.5f}".format('modelo_xgb_hp', folds, RMSE_xgb_hp.mean(), RMSE_xgb_hp.std()))
print('MAE:', mean_absolute_error(y_test1, xgb_hp_pred))
print('R2: ',  r2_score(y_test1, xgb_hp_pred) * 100)
```

```
#Aplico Cross-Validation para validar overfiting
xgb_hp_CV = cross_val_score(modelo_xgb_hp, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(xgb_hp_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_xgb_hp', '7', xgb_hp_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_xgb_hp', '7', xgb_hp_CV.std()))
```

```
# Predicción
xgb_hp_pred_log = modelo_xgb_hp.predict(X_test)
final_xgb_hp = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(xgb_hp_pred_log)})
final_xgb_hp.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_xgb_hp.csv", index=False)
```

Linear regression con hiperparametros - LASSO

```
from sklearn import linear_model

# tunning
modelo_lasso_hp = linear_model.Lasso()

param_grid = {
    'alpha': [0.00001, 0.001, 0.1, 1, 10, 50, 100],
    'max_iter': [100, 1000, 10000]
}

kfold = KFold(n_splits=folds, random_state=seed, shuffle=True)

scorer = make_scorer(rmse, greater_is_better=False)
```

```
grid_search = GridSearchCV(modelo_lasso_hp, param_grid, n_jobs=-1, cv=kfold, verbose=1, scoring=scorer)
grid_result = grid_search.fit(X_train1, y_train1)

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("{:06.5f} ({:06.5f}) with {}".format(mean, stdev, param))

# Resultados
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
# Model
modelo_lasso_hp = linear_model.Lasso(alpha= 0.001, max_iter=100)

# Fit & Predict
modelo_lasso_hp.fit(X_train1, y_train1)
lasso_hp_pred = modelo_lasso_hp.predict(X_test1)

#Gráfica
sns.set(font_scale=1.5)
plt.figure(figsize=(10, 5))
plt.scatter(y_test1, lasso_hp_pred, s=20)
plt.title('Predicted vs. Actual')
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')

plt.plot([min(y_test1), max(y_test1)], [min(y_test1), max(y_test1)])
plt.tight_layout()
```

```
RMSE_lasso_hp = rmse_cv(modelo_lasso_hp, X_train1, y_train1)

print("RMSE-{}-CV({})={:06.5f}+-
{:06.5f}".format('modelo_lasso_hp', folds, RMSE_lasso_hp.mean(), RMSE_lasso_hp.std()))
print('MAE:', mean_absolute_error(y_test1, lasso_hp_pred))
print('R2: ',  r2_score(y_test1, lasso_hp_pred) * 100)
```

```
#Aplico Cross-Validation para validar overfiting
lasso_hp_CV = cross_val_score(modelo_lasso_hp, X_train1, y_train1, scoring="neg_mean_squared_error", cv = 7)
print(lasso_hp_CV)
```

```
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_lasso_hp', '7', lasso_hp_CV.mean()))
print("Cross Validation-{}-CV({})={:06.5f}".format('modelo_lasso_hp', '7', lasso_hp_CV.std()))
```

```
# Predicción
lasso_hp_pred_log = modelo_lasso_hp.predict(X_test)
final_lasso_hp = pd.DataFrame({'Id':test_bis['Id'], 'SalePrice':np.expm1(lasso_hp_pred_log)})
final_lasso_hp.to_csv("/content/drive/MyDrive/Colab Notebooks/TFM/Data/final_lasso_hp.csv", index=False)
```

```
import joblib

modelo_final = modelo_RF_hp
joblib.dump(modelo_final, '/content/drive/MyDrive/Colab Notebooks/TFM/Data/modelo_RF_hp.pkl')
```

```
modelo_final = modelo_xgb
```

```
print(modelo_final)
```

Web Application

app_v2.py

```
from flask import Flask, render_template, url_for, request, redirect
import pandas as pd
import numpy as np
#import pickle
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
import joblib
from werkzeug.utils import secure_filename
from werkzeug.datastructures import  FileStorage
import os
from scipy import stats
from scipy.stats import norm, skew
from Funcion_Limpieza import limpieza, insertadb

app = Flask(__name__)

uploads_dir = 'C:/Users/lourd/Documents/Lulu/Cursos 2021/Master Data Science/22 Trabajo Fin de Master/Lulu API
1/instance/uploads'
os.makedirs(uploads_dir, exist_ok=True)

@app.route('/')


def home():
    return render_template('home.html')

@app.route('/predict', methods=['POST'])
def predict():


    modelo = open('modelo_RF_hp.pkl', 'rb')
    clf = joblib.load(modelo)


    if request.method == 'POST':
        message = request.files['fileupload']
        message.save(os.path.join(uploads_dir, secure_filename(message.filename)))
        data = pd.read_csv(message.filename)
        df_id = data["Id"]

        datos = limpieza(data)

        my_prediction = np.expm1(clf.predict(datos))

        df_id_b = pd.DataFrame(df_id, columns=['Id'])
        df_mypred = pd.DataFrame(my_prediction, columns=['pred'])
        df_id_bi = pd.concat([df_id_b, df_mypred], axis=1)
        df_id_b = df_id_bi.to_numpy()

        insertar = insertadb(df_id_bi)


    return render_template('result.html', prediction=my_prediction, tam=df_id_b)


if __name__ == '__main__':
   app.run(debug=True)
```

Funcion_Limpieza.py

```
import pandas as pd
import numpy as np
from scipy import stats
from scipy.stats import norm, skew
import mysql.connector

def limpieza(data):


    #PRE-PROCESSING & DATA CLEANING
        train = pd.read_csv('train.csv')
        df_id = data["Id"]
        df_id = df_id.to_numpy()
        train = train.drop(["SalePrice"], axis=1)
        junto_df = pd.concat([train, data], ignore_index=True)

        #Manejo de categoricas
        categoricas = [col for col in junto_df.columns.values if junto_df[col].dtype == 'object']
        categoricas_df = junto_df[categoricas]
        numericas_df = junto_df.drop(categoricas, axis=1)

        #Manejo de asimetria para numericas
        num_skew = numericas_df.apply(lambda x: skew(x.dropna()))
        num_skew = num_skew[num_skew > 0.75]
        numericas_df[num_skew.index] = np.log1p(numericas_df[num_skew.index])

        #Manejo de missing values
        data_len = numericas_df.shape[0]

        for col in numericas_df.columns.values:
            missing_values = numericas_df[col].isnull().sum()

            # drop si tiene mas de 50 valores valores missing
            if missing_values > 50:
                numericas_df = numericas_df.drop(col, axis=1)
            # Si es menor a 50, lo lleno con la media
            else:
                numericas_df = numericas_df.fillna(numericas_df[col].median())


        data_len = categoricas_df.shape[0]

        for col in categoricas_df.columns.values:
            missing_values = categoricas_df[col].isnull().sum()

            # drop si tiene mas de 50 valores missing
            if missing_values > 50:
                print("Eliminando columna: {}".format(col))
                categoricas_df = categoricas_df.drop(col, axis=1)

            # Si es menor a 50, lo lleno con 'No Aplica'
            else:
                categoricas_df = categoricas_df.fillna('No aplica')
                pass

        categoricas_df_dummies = pd.get_dummies(categoricas_df)
        datos = pd.concat([numericas_df, categoricas_df_dummies], axis=1)
        datos = datos.drop(["Id"], axis=1)
        datos = datos.iloc[len(train):]

        #FIN PRE-PROCESSING & DATA CLEANING
```

```
        #INSERTO EN BASE DE DATOS


        miConexion = mysql.connector.connect(host='localhost', user='root', passwd='root', db='modelo_casas')
        cur = miConexion.cursor()

        #train = pd.read_csv('test_carga_mysql.csv', index_col=False, delimiter=',')
        train = data
        train = train.fillna(0)

        for i, row in train.iterrows():
            sql = "INSERT INTO modelo_casas.casas VALUES
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%
s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,
%s,%s,%s,%s,%s,%s,%s,0,0)"
            cur.execute(sql, tuple(row))
            miConexion.commit()

        miConexion.close()

        #FIN DE INSERTAR EN BASE DE DATOS

        return datos


def insertadb(df_id_b):
    miConexion = mysql.connector.connect(host='localhost', user='root', passwd='root', db='modelo_casas')
    cur = miConexion.cursor()

    for i, row in df_id_b.iterrows():
        sql = "UPDATE modelo_casas.casas SET Prediccion = %s WHERE ID = %s"
        cur.execute(sql, (row['pred'], row['Id']))
        miConexion.commit()

    miConexion.close()

    return
```

.