Hasta ahora en el código se han hecho los siguientes procedimientos:

- Las columnas 'Alley', 'FireplaceQu', 'PoolQC', 'Fence' y 'MiscFeature' tienen más del 30% de la data nula han sido **eliminadas** por tener más del 30% de data nula.
- Se han **reemplazado** los valores nulos de la columna 'MiscFeature', que mide en sí los elementos extra que tiene el inmueble, con valores 0.
- Se ha **eliminado** la columna 'GarageCars' por estar **muy correlacionada** con la variable GarageArea en sí.
- Hemos **reemplazado** las variables categóricas que tenían valores nulos con su valor más frecuente.
- Se ha **reemplazado** las variables numéricas, **randomizando** entre rangos específicos y poniendo los límites por si el reemplazo alcanza un mínimo negativo.
- Se corrige el outlier con índice 1132, en donde el año estaba mal.
- Se ha **reemplazado** las variables categóricas por números (se identifican las variables categóricas como columnas del tipo objeto) Esto puede ser reemplazado con TargetEncoding?
- Se sacarán las dummies de las variables categóricas.
- Finalmente, se **separa** la data de entrenamiento y prueba en X e Y para poder hacer la medición del modelo.

Lectura del kernel:
Stacked Regressions : Top 4% on LeaderBoard

Algunos temas nuevos que se ven en este kernel son:

- Box-Cox Transformation: Es un método para **normalizar** las variables dependientes (Y) según el paper de Box-Cox. Juega con un parámetro ƛ, y puedes ver las especificaciones acá.
- Robustidad hacia los outliers: **RobustScaler()** La introducción a este kernel especifica que se ha tenido como goal hacer los modelos LB y Xgboost robustos a outliers. Esto porque, envés de eliminar todos los outliers, hacemos robusto todo el modelo en sí a los outliers.
- QQ-plot: Conocido en español como gráfico cuantil-cuantil, es un tipo de gráfico que nos permite comparar la distribución de dos muestras. Es decir, si queremos saber si la muestra tiene una distribución normal, podemos observar el gráfico qq y aceptar o rechazar la Hipótesis nula. Podemos ver una explicación bien chévere acá. Básicamente, el criterio es que todos los puntos del qqplot deben estar en la línea que se dibuja.

Básicamente el pipeline del kernel es:

1. Eliminar outliers. Haciendo un rápido scatter plot de GrLivArea con SalePrice, se puede ver que hay 2 outliers (que indican que un altísimo GrLivArea y muy bajo SalePrice). Entonces, delimita el GrLivArea en un rango específico. (GrLivArea mayor que 4000 y SalePrice menor a 300mil)
2. Al observar que la variable Y no está distribuida de manera normal, le aplica el logaritmo + 1 con la función np.log1p(train['SalePrice]) para normalizarla.
3. Reemplaza los valores nulos. No elimina 'PoolQC', 'Alley', 'Fence', 'FirePlaceQu', 'MiscFeature' sino los reemplaza por "None" que significa que la casa no tiene esas características. **Diferente a mi modelo**
4. Reemplaza los nulos de LotFrontAge por la media de la data **agrupada** por barrio (neighborhood). Básicamente porque el área de la calle se relaciona con el LotFrontage.

5. Reemplaza los nulos numéricos con cero. Hace algunas observaciones en diferentes features. Por ejemplo, **en 'Utilities' se da cuenta de que todos los valores son en sí** "AllPub", exceptuando unos pocos. Entonces elimina la columna.
6. Transforma algunos valores numéricos que en sí son categóricos. Columnas como 'MSSubClass', 'OverallCond','YrSold','MoSold' <- analizar si afecta a mi kernel.
7. Usa la función **LabelEncoder** para transformar algunas variables categóricas.
8. Crea una variable 'TotalSF' que nos dará el área total construida? osea, la suma de las áreas 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF'
9. Luego saca el Skew (asimetría) de todas las variables que no son objects y a estas variables les **aplica la transformación BoxCox**, sin embargo, no optimiza el parámetro lambda y solo usa lambda = 1.5 para todas las variables transformadas. Qué pasaría si lo optimizo para todas las variables?
10. Saca dummies para las variables categóricas.

Para modelar hace los siguientes pasos:

1. Le agrega el shuffle a la función cross_val_score para poder hacer luego el **cross-validation. (Profundizar en el método)** Crea una función para entrenar directamente y hacer el shuffle de cross validation, así solo tendrá que llamar a los modelos en la función para que pueda ver el entrenamiento.
2. Hace pipelines para los modelos Lasso y ElasticNet, agregándole el método **RobustScaler()**. Para hacerlos robustos a outliers.
3. Crea los objetos para los modelos: KernelRidgeRegression, GradientBoostRegression, XGBoost y LightGBM.
4. Le hace **Stacking básico** a los modelos. Para esto, crea una clase de AveragingModels que combinará los distintos modelos. Según él, dice que es la forma más fácil de hacer el stacking.
5. Hará un **Stacking intermedio** a los modelos, poniendo un meta-model. Esto lo explicaremos en el siguiente cuadro:
   caveat: El código se va a ir corriendo mientras se va aprendiendo.

Para hacer el stacking con un modelo meta. Tendremos que hacer los siguientes pasos.

1. Dividir la data de entrenamiento en dos partes: train y holdout
2. Testear la data entrenada con train en holdout.
3. Usar las predicciones hechas para el holdout y entrenar un modelo en base a eso.
   A este proceso se le llama el k-fold-Stacking.