



Laboratorio de proyecto

House Prices - Advanced Regression Techniques

Competición Kaggle #1



Punto de partida y objetivos

- ❑ **Data set** con datos de viviendas
- ❑ **79 variables** diferentes
- ❑ Procesamiento de datos e **ingeniería de variables**
- ❑ Aplicar técnicas de **regresión**
- ❑ **Predecir** precios de venta de otro data set



Razones de la elección:

- ☐ Reto sencillo y similar a otros problemas resueltos durante el Máster
- ☐ Aprender a procesar datos
- ☐ Resultados tangibles y fáciles de asimilar: predicción de precios



Proceso de resolución del problema

- ❑ **Importar librerías**, definir constantes, montar drive y cargar datos.
- ❑ **Análisis de variables**, tanto en train como en test datasets.
- ❑ **Entrenar modelo**.
- ❑ Aplicación de modelo al dataset test. **Predicción**.
- ❑ Carga y **evaluación de resultados en Kaggle**.



Análisis de variables

- ❑ Averiguamos qué columnas tienen **valores nulos**.
- ❑ Comprobamos qué columnas tienen **datos numéricos**.
- ❑ Analizamos estas columnas y obtenemos el **coeficiente de correlación de Spearman** con la columna SalePrice.



```
1 # Extracción de datos agregados  
2 data[columnas_numericas].corr(method='spearman')['SalePrice']
```

Id	-0.018546
MSSubClass	0.007192
LotFrontage	0.409076
LotArea	0.456461
OverallQual	0.809829
OverallCond	-0.129325
YearBuilt	0.652682



Análisis de variables

- ❑ Identificar columnas con **datos categóricos**.
- ❑ De estas columnas **extraemos datos agregados** y así sabemos cuantos valores diferentes hay en cada una, cual es el que mas se repite...

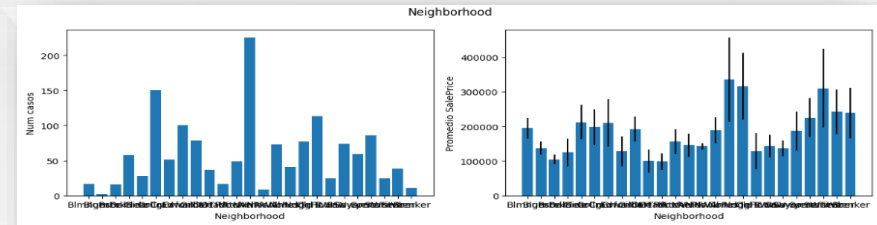
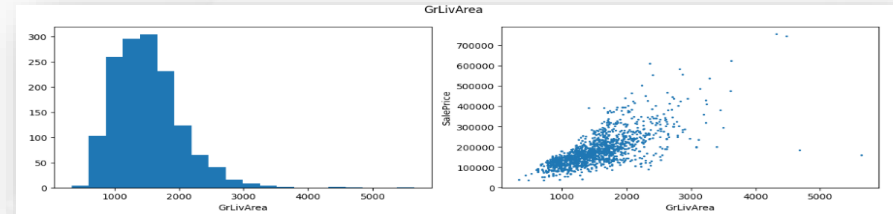
```
[ ] 1 # Identificar columnas categóricas
     2 columnas_categoricas = data.columns[data.dtypes == "object"]
     3
     4 # Extracción de datos agregados
     5 data[columnas_categoricas].describe().transpose()
```

	count	unique	top	freq
MSZoning	1460	5	RL	1151
Street	1460	2	Pave	1454
Alley	91	2	Grvl	50
LotShape	1460	4	Reg	925
LandContour	1460	4	Lvl	1311
Utilities	1460	2	AllPub	1459



Análisis de variables

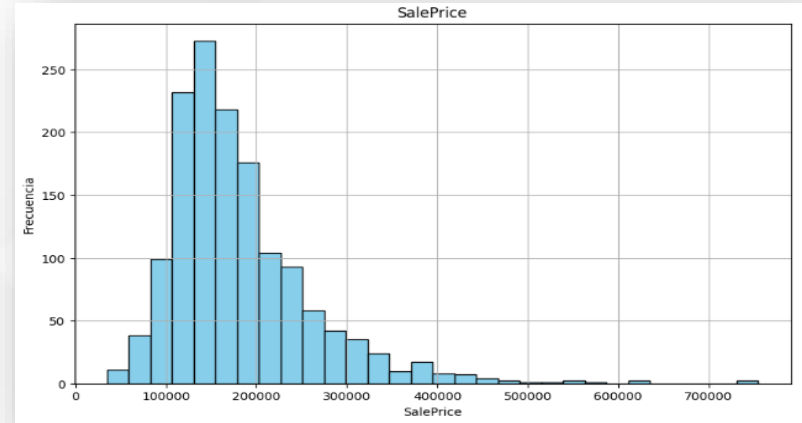
□ **Representación gráfica de variables numéricas y categóricas** (distribución y relación con variable objetivo), **para intentar identificar patrones.**





Análisis de variables

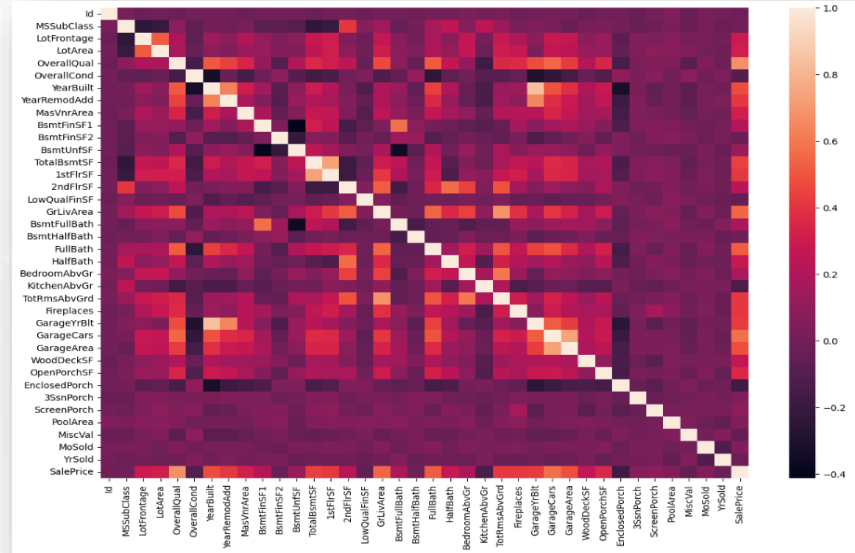
- Representación gráfica de la distribución de la variable objetivo.





Análisis de variables

❑ Buscamos
correlación entre
variables.





Análisis de variables

❑ **Eliminación de variables** para aligerar el modelo sin perder información. Nos basamos en:

- Diagramas de barras – **variables con desviación estándar muy alta**
- Correlación – eliminar **variables de pares con alta correlación**
- **ID, no aporta** nada a la predicción
- Rellenar NaNs ????



Análisis de variables

- ❑ **Transformación de variables categóricas** con los siguientes métodos:
 - **Label encoding**
 - Para variables que se miden en grados de calidad
 - Para otras variables categóricas
 - **One hot encoding** – para variables “Miscellaneous”
 - **Target encoding** – para variables que tienen alta relación con la variable objetivo



Entrenar modelo

- ❑ Separación de la variable objetivo

```
1 X = processed_data.drop(columns=["SalePrice"], inplace=False)
2 y = processed_data.SalePrice
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=None, random_state=0)
```

- ❑ Utilizamos **validación cruzada** para ajustar modelos. En nuestro caso dividimos los datos en 5 conjuntos

```
1 # Dividimos los datos de entrenamiento en 5 partes
2 shuffle_split = ShuffleSplit(n_splits=5, random_state=0)
```

- ❑ Decidimos entrenar nuestro modelo con el algoritmo **XGBoost**, ya que da buenos resultados en predicciones de regresiones logísticas.

Entrenar modelo

- ❑ Definimos la **mallade hiperparámetros**, dando varios valores a cada variable.
- ❑ Definimos modelo para entrenar mediante **validación cruzada con búsqueda en mallade hiperparámetros**.
- ❑ Elegimos validar con el error cuadrático medio.

```
1 # dict_params = {"n_estimators": [200, 250, 300],
2 #               "max_depth": [15, 20, 25],
3 #               "learning_rate": [0.03, 0.06],
4 #               "objective": ["reg:squarederror"],
5 #               "tree_method": ["hist"],
6 #               "subsample": [0.6],
7 #               }
```

```
from sklearn.model_selection import GridSearchCV
model_XGB = GridSearchCV(estimator=XGBRegressor(random_state=42997296),
                        param_grid=dict_params,
                        cv=shuffle_split,
                        refit=True,
                        verbose=1,
                        n_jobs=-1,
                        return_train_score=True,
                        scoring="neg_mean_squared_error"
                        )
```

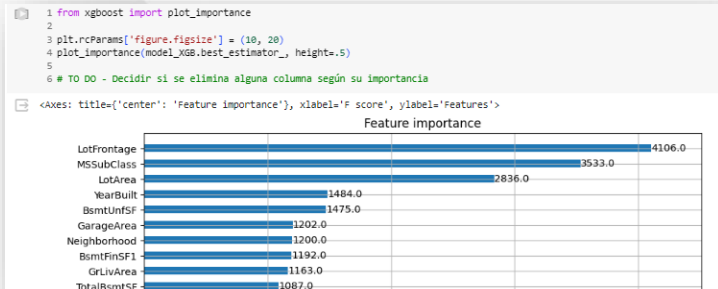
Entrenar modelo

- Entrenamos nuestro modelo.

```
1 model_XGB.fit(X_train, y_train)
2 print(model_XGB.best_params_)
```

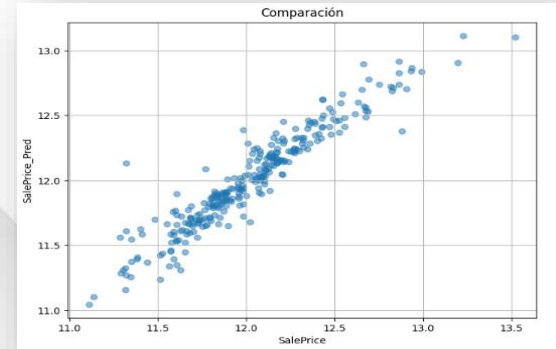
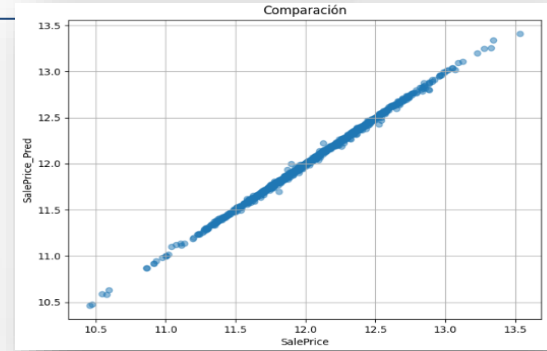
```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
{'learning_rate': 0.03, 'max_depth': 20, 'n_estimators': 300, 'objective': 'reg:squarederror', 'subsample': 0.6, 'tree_method': 'hist'}
```

- Obtenemos cuáles son los mejores hiperparámetros.
- Observamos también la **importancia de cada columna**, para ver una vez más si podemos eliminar alguna, cosa que no hacemos.



Predicción

- ❑ **Aplicamos nuestro mejor modelo**, el cual hemos guardado previamente, a los datos de test, para obtener la predicción de la variable objetivo.
- ❑ **Analizamos los resultados** de la predicción con los datos de entrenamiento y de validación.



Predicción

- ❑ **Obtenemos la predicción con los datos de test.**
- ❑ **Almacenamos los datos en un archivo .CSV.**

```
1 preds = model_XGB.predict(processed_test_data)
2 preds = np.exp(preds)
3 preds

array([[124750.23, 161052.52, 186203.92, ..., 164282.72, 104076.87,
       201030.16], dtype=float32)


1 # Submission
2 test_data = pd.read_csv(os.path.join(rootdir, 'test.csv'))
3 ids = test_data.pop('Id')
4
5 output = pd.DataFrame({'Id': ids,
6                        'SalePrice': preds})


1 from datetime import datetime
2
3 now = datetime.now()
4 dt_string = now.strftime("%Y%m%d_%H%M%S")
5
6 output.to_csv( os.path.join(rootdir, 'submission_xgboost_'+dt_string+'_log_y.csv') , index=False)
```



Evaluación

- ☐ Almacenamos los datos en un archivo .csv y lo enviamos a kaggle.
- ☐ Obtenemos una puntuación de **0,278**.
- ☐ Después de esto, intentamos mejorar nuestro modelo, haciendo una **predicción del valor logarítmico del precio**.
- ☐ Repetimos proceso y obtenemos puntuación de **0,139**, quedándonos en la posición **1927** / 4873.

1927	Raúl Garcia Martinez		0.13948	1	6m
------	----------------------	---	---------	---	----



Your Best Entry!
Your most recent submission scored 0.13948, which is the same as your previous score. Keep trying!

Store Sales – Time Series Forecasting

Competición Kaggle #2



Punto de partida y objetivos:

❑ Varios datasets:

- Ventas en tiendas
- Tipos y datos de tiendas
- Precio petróleo
- Vacaciones
- Generar un modelo de predicción de ventas sobre otro dataset



Razones de la elección:

- ❑ Profundizar en el **procesado de los datos**
- ❑ Aprender a usar algoritmos para **predecir datos de series temporales**
- ❑ Similar a **casos de uso reales**:
 - Venta de billetes de avión
 - Venta de entradas



Proceso de resolución del problema

- ☐ **Importar librerías**, definir constantes, montar drive y **cargar datos**.
- ☐ **Análisis de variables**, tanto en train como en test datasets:
- ☐ **Integrar datasets** en uno único.
- ☐ **Codificación** de variables.
- ☐ **Entrenar modelo**.
- ☐ Aplicación de modelo al dataset test. **Predicción**.
- ☐ Carga y **evaluación de resultados** en Kaggle.

Carga de datos

- ❑ Cargamos y leemos datos de los distintos datasets: train, test, transactions, stores, oil y holidays.

```
1 #Leemos las primeras filas de cada dataset
2 data.head()
```

	id	date	store_nbr	family	sales	onpromotion
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0
1	1	2013-01-01	1	BABY CARE	0.0	0
2	2	2013-01-01	1	BI	1	stores.head()
3	3	2013-01-01	1	BEVER		
4	4	2013-01-01	1	E		

	store_nbr	city	state	type	cluster
0	1	Quito	Pichincha	D	13
1	2	Quito	Pichincha	D	13
2	3	Quito	Pichincha	D	8
3	4	Quito	Pichincha	D	9
4	5	Santo Domingo	Santo Domingo de los Tsachilas	D	4

```
1 transactions.head()
```

	date	store_nbr	transactions
0	2013-01-01	25	770
1	2013-01-02	1	2111
2	2013-01-02	2	
3	2013-01-02	3	
4	2013-01-02	4	

```
1 oil.head()
```

	date	dcoilwtico
0	2013-01-01	NaN
1	2013-01-02	93.14
2	2013-01-03	92.97

```
1 holidays_events
```

	date	type	locale	locale_name	description	transferred
0	2012-03-02	Holiday	Local	Manta	Fundacion de Manta	False
1	2012-04-01	Holiday	Regional	Cotopaxi	Provincializacion de Cotopaxi	False
2	2012-04-12	Holiday	Local	Cuenca	Fundacion de Cuenca	False
3	2012-04-14	Holiday	Local	Libertad	Cantonizacion de Libertad	False
4	2012-04-21	Holiday	Local	Riobamba	Cantonizacion de Riobamba	False



Análisis de variables

- ❑ Comprobamos qué datasets contienen **datos nulos**.
- ❑ Vemos que el precio del petróleo solo contiene datos de lunes a viernes. **Interpolamos los valores del viernes y el sábado** para tener una serie temporal continua.
- ❑ El primer valor no puede interpolarse y lo copiamos del día siguiente.

	date	dcoilwtico
0	2013-01-01	NaN
1	2013-01-02	93.14
2	2013-01-03	92.97
3	2013-01-04	93.12
4	2013-01-07	93.14

	date	dcoilwtico
	2013-01-01	93.140000
	2013-01-02	93.140000
	2013-01-03	92.970000
	2013-01-04	93.120000
	2013-01-05	93.146667





Análisis de variables

❑ Vemos que hay tiendas que abrieron más tarde que el inicio de la serie temporal.

❑ Eliminamos datos de ventas de esas tiendas.

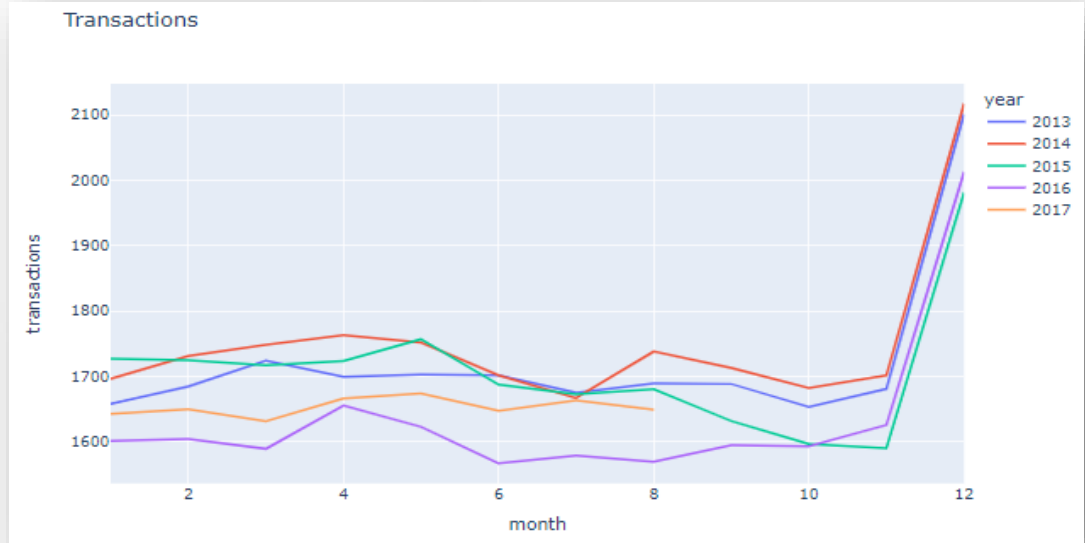
```
1 # Quitamos datos de ventas de algunas tiendas que abrieron más tarde del inicio de la serie temporal
2
3 print(data_processed.shape)
4 data_processed = data_processed[~((data_processed.store_nbr == 52) & (data_processed.date < "2017-04-20"))]
5 data_processed = data_processed[~((data_processed.store_nbr == 22) & (data_processed.date < "2015-10-09"))]
6 data_processed = data_processed[~((data_processed.store_nbr == 42) & (data_processed.date < "2015-08-21"))]
7 data_processed = data_processed[~((data_processed.store_nbr == 21) & (data_processed.date < "2015-07-24"))]
8 data_processed = data_processed[~((data_processed.store_nbr == 29) & (data_processed.date < "2015-03-20"))]
9 data_processed = data_processed[~((data_processed.store_nbr == 20) & (data_processed.date < "2015-02-13"))]
10 data_processed = data_processed[~((data_processed.store_nbr == 53) & (data_processed.date < "2014-05-29"))]
11 data_processed = data_processed[~((data_processed.store_nbr == 36) & (data_processed.date < "2013-05-09"))]
12 data_processed.shape

(3000888, 6)
(2780316, 6)
```



Análisis de variables

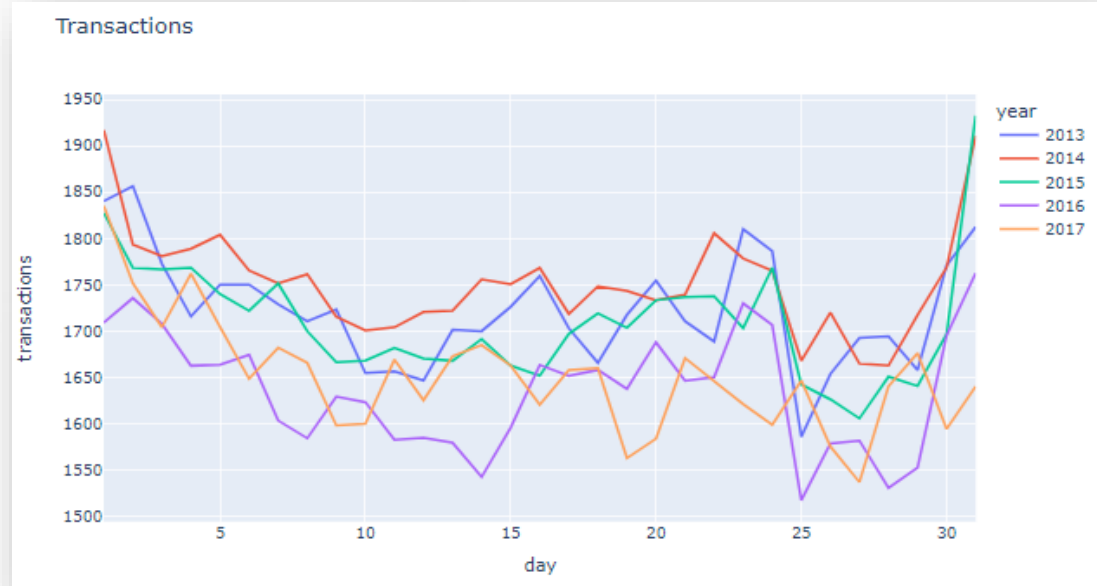
❑ Estudiamos el comportamiento del número de transacciones en diferentes periodos de tiempo.





Análisis de variables

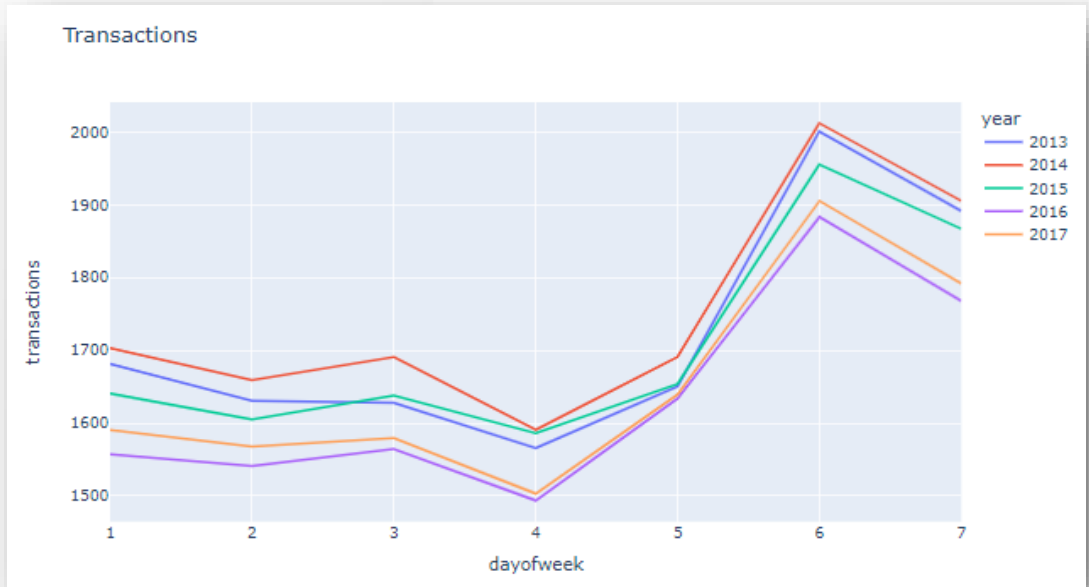
□ Estudiamos el comportamiento del número de transacciones en diferentes periodos de tiempo.





Análisis de variables

❑ Estudiamos el comportamiento del número de transacciones en diferentes periodos de tiempo.





Análisis de variables

- ❑ Vamos a generar nuevas variables en el dataset: **mes**, **día** y **día de la semana**:

```
1 data_processed.sample(100)
```

	id	date	store_nbr	family	sales	onpromotion	month	day	day_of_week
129956	129956	2013-03-14	6	BEAUTY	1.000	0	3	14	3
1742549	1742549	2015-09-07	51	HOME APPLIANCES	0.000	0	9	7	0
1352521	1352521	2015-01-31	9	HOME AND KITCHEN II	23.000	0	1	31	5
1940706	1940706	2015-12-29	12	DELI	237.000	0	12	29	1
1667429	1667429	2015-07-27	44	BREAD/BAKERY	1088.555	2	7	27	0
...
1638290	1638290	2015-07-11	27	BREAD/BAKERY	627.833	0	7	11	5
2934498	2934498	2017-07-09	46	CELEBRATION	17.000	0	7	9	6
2137141	2137141	2016-04-17	23	POULTRY	398.283	1	4	17	6
1652913	1652913	2015-07-19	37	DELI	174.870	1	7	19	6



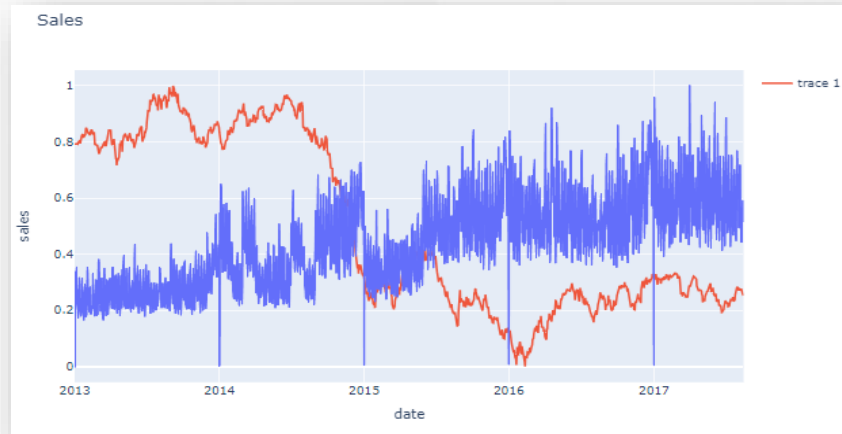
Análisis de variables

- ❑ Introducimos otra nueva variable: **media móvil** de ventas en los último 30 días

id	date	store_nbr	family	onpromotion	month	day	day_of_week	sales_media_movil_30_dias_lag16
3000888	2017-08-16	1	AUTOMOTIVE	0	8	16	2	2.500000
3000889	2017-08-16	1	BABY CARE	0	8	16	2	10.133333
3000890	2017-08-16	1	BEAUTY	2	8	16	2	8.766667
3000891	2017-08-16	1	BEVERAGES	20	8	16	2	1.033333
3000892	2017-08-16	1	BOOKS	0	8	16	2	4.633333
...
...	2017-							

✦ Análisis de variables

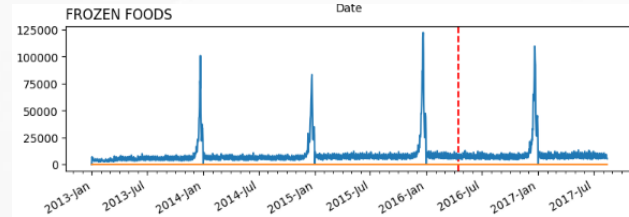
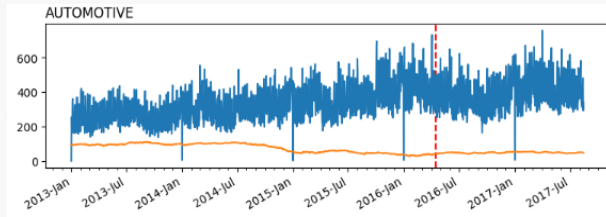
- ❑ Generamos distintas gráficas para ver si podemos **observar algún patrón** en las ventas.
- ❑ Precio petróleo vs Ventas



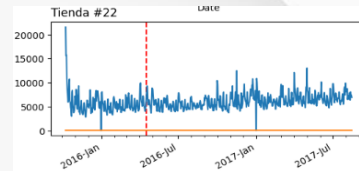
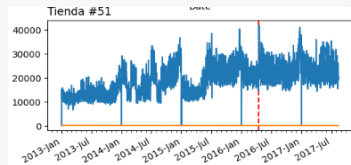
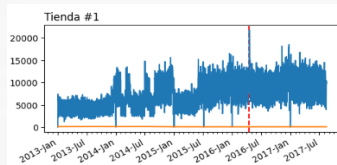


Análisis de variables

- ☐ Generamos distintas gráficas para ver si podemos observar algún patrón en las ventas.
- ☐ Ventas por familias



- ☐ Ventas por tiendas





Análisis de variables

- ☐ Vemos que hay algunas gráficas en las que algunos patrones se repiten.
- ☐ Nada tan concluyente como para procesar datos de alguna otra manera.
- ☐ Conclusión: continuamos con el dataset con los datos completos.

Combining datasets

❑ **Añadir datos de tiendas:** merge data / test y stores a partir del número de tienda

```
5 data_processed = pd.merge(data_processed, stores_processed, on="store_nbr", how="left")
6 data_processed.head()
```

	id	date	store_nbr	family	sales	onpromotion	month	day	day_of_week	sales_media_movil_30_dias_lag16	city	state	type	cluster
0	0	2013-01-01	1	AUTOMOTIVE	0.0	0	1	1	1	NaN	Quito	Pichincha	D	13
1	1	2013-01-01	1	BABY CARE	0.0	0	1	1	1	4.666667	Quito	Pichincha	D	13
2	2	2013-01-01	1	BEAUTY	0.0	0	1	1	1	0.000000	Quito	Pichincha	D	13
3	3	2013-01-01	1	BEVERAGES	0.0	0	1	1	1	3.700000	Quito	Pichincha	D	13
4	4	2013-01-01	1	BOOKS	0.0	0	1	1	1	2227.433333	Quito	Pichincha	D	13

```
1 test_processed = pd.merge(test_processed, stores_processed, on="store_nbr", how="left")
2 test_processed.head()
```

	id	date	store_nbr	family	onpromotion	month	day	day_of_week	sales_media_movil_30_dias_lag16	city	state	type	cluster
0	3000888	2017-08-16	1	AUTOMOTIVE	0	8	16	2	2.500000	Quito	Pichincha	D	13
1	3000889	2017-08-16	1	BABY CARE	0	8	16	2	10.133333	Quito	Pichincha	D	13
2	3000890	2017-08-16	1	BEAUTY	2	8	16	2	8.766667	Quito	Pichincha	D	13
3	3000891	2017-08-16	1	BEVERAGES	20	8	16	2	1.033333	Quito	Pichincha	D	13
4	3000892	2017-08-16	1	BOOKS	0	8	16	2	4.633333	Quito	Pichincha	D	13

Combining datasets

- ❑ **Añadir datos del precio del petróleo: merge data / test y oil a partir de la fecha de la transacción**

```
3 if 'dcoilwtico' not in data_processed.columns:
4     data_processed = pd.merge(data_processed, oil_processed, on="date", how="left")
5 data_processed.head()
6
7 if 'dcoilwtico' not in test_processed.columns:
8     test_processed = pd.merge(test_processed, oil_processed, on="date", how="left")
9 test_processed.head()
```

	id	date	store_nbr	family	onpromotion	month	day	day_of_week	sales_media_movil_30_dias_lag16	city	state	type	cluster	dcoilwtico
0	3000888	2017-08-16	1	AUTOMOTIVE	0	8	16	2	2.500000	Quito	Pichincha	D	13	46.8
1	3000889	2017-08-16	1	BABY CARE	0	8	16	2	10.133333	Quito	Pichincha	D	13	46.8
2	3000890	2017-08-16	1	BEAUTY	2	8	16	2	8.766667	Quito	Pichincha	D	13	46.8
3	3000891	2017-08-16	1	BEVERAGES	20	8	16	2	1.033333	Quito	Pichincha	D	13	46.8
4	3000892	2017-08-16	1	BOOKS	0	8	16	2	4.633333	Quito	Pichincha	D	13	46.8

✦ Combinar datasets

❑ Añadir datos de vacaciones:

- Primero quitamos la información de vacaciones anteriores a la fecha de inicio del dataset de ventas.
- Incorporamos los datos de vacaciones type y description mediante un bucle que recorre todo el dataset.
- **Condiciones:**
 - Si en dataset holidays, el valor type es locale, y el nombre de la ciudad coincide con la ciudad en el dataset data
 - Si en dataset holidays, el valor type es regional, y el nombre del estado coincide con la estado en el dataset data
 - Si en dataset holidays, el valor type es nacional

Combinar datasets

❑ Añadir datos de **vacaciones**:

	id	date	store_nbr	family	onpromotion	month	day	day_of_week	sales_media_movil_30_dias_lag16	city	state	type	cluster	dcoilwtico	holiday_locale	holiday_type
8835	3009723	2017-08-20	7	MEATS	0	8	20	6	0.000000	Quito	Pichincha	D	8	47.790000	NaN	NaN
9669	3010557	2017-08-21	30	AUTOMOTIVE	0	8	21	0	137.200000	Guayaquil	Guayas	C	3	47.390000	NaN	NaN
15757	3016645	2017-08-24	50	HOME AND KITCHEN II	2	8	24	3	0.000000	Ambato	Tungurahua	A	14	47.240000	Local	Holiday
2549	3003437	2017-08-17	30	DAIRY	11	8	17	3	167.200000	Guayaquil	Guayas	C	3	47.070000	NaN	NaN
23824	3024712	2017-08-29	27	SCHOOL AND OFFICE SUPPLIES	0	8	29	1	54.015233	Daule	Guayas	D	1	46.460000	NaN	NaN
19467	3020355	2017-08-26	54	PRODUCE	0	8	26	5	2.900000	El Carmen	Manabi	C	3	47.233333	NaN	NaN
21489	3022377	2017-08-28	12	CELEBRATION	0	8	28	0	0.000000	Latacunga	Cotopaxi	C	15	46.400000	NaN	NaN
17025	3017913	2017-08-25	36	PRODUCE	1	8	25	4	2.966667	Libertad	Guayas	E	10	47.650000	NaN	NaN
20059	3020947	2017-08-27	21	POULTRY	0	8	27	6	21.566667	Santo Domingo	Santo Domingo de los Tsachilas	B	6	46.816667	NaN	NaN
17700	3018588	2017-08-25	6	GROCERY I	48	8	25	4	0.133333	Quito	Pichincha	D	13	47.650000	NaN	NaN

❑ Añadir el evento del **terremoto**

✦ Codificar variables

- ❑ Una vez que hemos incorporado todos los datos en un solo dataset, procedemos a **codificar las variables categóricas**.
- ❑ Primero leemos cuántos valores distintos existen en las columnas con valores categóricos:

```
['AUTOMOTIVE' 'BABY CARE' 'BEAUTY' 'BEVERAGES' 'BOOKS' 'BREAD/BAKERY'  
'CELEBRATION' 'CLEANING' 'DAIRY' 'DELI' 'EGGS' 'FROZEN FOODS' 'GROCERY I'  
'GROCERY II' 'HARDWARE' 'HOME AND KITCHEN I' 'HOME AND KITCHEN II'  
'HOME APPLIANCES' 'HOME CARE' 'LADIESWEAR' 'LAWN AND GARDEN' 'LINGERIE'  
'LIQUOR,WINE,BEER' 'MAGAZINES' 'MEATS' 'PERSONAL CARE' 'PET SUPPLIES'  
'PLAYERS AND ELECTRONICS' 'POULTRY' 'PREPARED FOODS' 'PRODUCE'  
'SCHOOL AND OFFICE SUPPLIES' 'SEAFOOD']  
['Quito' 'Cayambe' 'Latacunga' 'Riobamba' 'Ibarra' 'Santo Domingo'  
'Guaranda' 'Ambato' 'Guayaquil' 'Salinas' 'Daule' 'Babahoyo' 'Quevedo'  
'Playas' 'Cuenca' 'Loja' 'Machala' 'Esmeraldas' 'El Carmen' 'Libertad'  
'Manta' 'Puyo']  
['Pichincha' 'Cotopaxi' 'Chimborazo' 'Imbabura'  
'Santo Domingo de los Tsachilas' 'Bolívar' 'Tungurahua' 'Guayas'  
'Santa Elena' 'Los Rios' 'Azuay' 'Loja' 'El Oro' 'Esmeraldas' 'Manabi'  
'Pastaza']  
['D' 'C' 'B' 'E' 'A']  
['National' nan 'Regional' 'Local']  
['Holiday' nan 'Work Day' 'Additional' 'Event' 'Transfer' 'Bridge']  
[0 1]
```



Codificar variables

- ❑ En las columnas category, city y state existen **muchos valores diferentes** ❑ normalizamos mediante **target encoding**
- ❑ En la columna type existen **sólo 5 valores diferentes** en ambos datasets (train / test) ❑ normalizamos mediante **one-hot encoding**, generando nuevas columnas
- ❑ En las columnas relativas a los datos de vacaciones vemos que hay **distinto número de valores diferentes** ❑ normalizamos mediante **label encoding** en lugar de one-hot, para mantener el mismo número de columnas en ambos datasets
- ❑ En las columnas de los datos de vacaciones, se han generado valores nulos en aquellas filas cuya fecha no es vacaciones. Sustituimos NaN por valores 0.

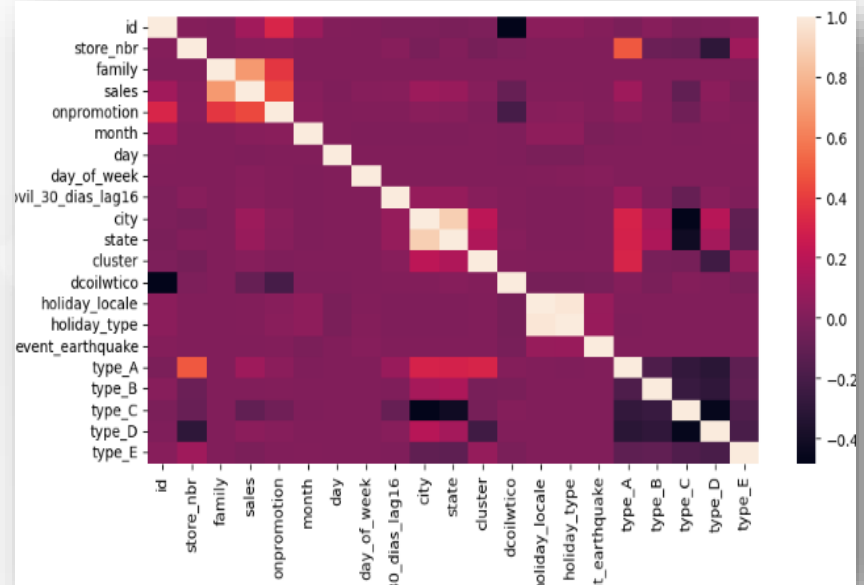
Codificar variables

❑ Resultado:

	id	date	store_nbr	family	sales	onpromotion	month	day	day_of_week	sales_media_movil_30_dias_lag16	city	state	cluster	dcoilwtico	holiday_locale	holiday_type	event_earthquake	family_name	type_A	type_B	type_C	type_D	type_E
0	0	2013-01-01	1	6.518422	0.000	0	1	1	1	0.000000	565.589351	562.317708	13	93.14	1	1	0	AUTOMOTIVE	0	0	0	1	0
1	1	2013-01-01	1	0.118006	0.000	0	1	1	1	4.666667	565.589351	562.317708	13	93.14	1	1	0	BABY CARE	0	0	0	1	0
2	2	2013-01-01	1	3.969794	0.000	0	1	1	1	0.000000	565.589351	562.317708	13	93.14	1	1	0	BEAUTY	0	0	0	1	0
3	3	2013-01-01	1	2548.927182	0.000	0	1	1	1	3.700000	565.589351	562.317708	13	93.14	1	1	0	BEVERAGES	0	0	0	1	0
4	4	2013-01-01	1	0.075638	0.000	0	1	1	1	2227.433333	565.589351	562.317708	13	93.14	1	1	0	BOOKS	0	0	0	1	0
...
2780311	3000883	2017-08-15	9	374.500734	438.133	0	8	15	1	2.633333	565.589351	562.317708	6	47.57	0	0	0	POULTRY	0	1	0	0	0
2780312	3000884	2017-08-15	9	103.387085	154.553	1	8	15	1	7.633333	565.589351	562.317708	6	47.57	0	0	0	PREPARED FOODS	0	1	0	0	0
2780313	3000885	2017-08-15	9	1441.617142	2419.729	148	8	15	1	596.142265	565.589351	562.317708	6	47.57	0	0	0	PRODUCE	0	1	0	0	0
2780314	3000886	2017-08-15	9	3.164105	121.000	8	8	15	1	77.572233	565.589351	562.317708	6	47.57	0	0	0	SCHOOL AND OFFICE SUPPLIES	0	1	0	0	0

✦ Codificar variables

- ❑ Comprobamos que todos los valores son de tipo numérico o fecha.
- ❑ Comprobamos **correlación entre variables**, una vez que todos los datos están incorporados y normalizados:



Codificar variables

- ☐ **No se observa correlación entre variables**, por lo que no podemos eliminar ninguna de nuestros dataset.
- ☐ Guardamos datasets combinados para que su carga sea más rápida.



Entrenar modelo

- ❑ Elegimos **Prophet**, algoritmo diseñado por Meta, como algoritmo de entrenamiento, ya que en la tutoría, el profesor nos lo presentó como **uno de los que mejor funcionan para predicciones con series temporales**.
 - Cargamos dataset previamente procesado y guardado
 - Instalamos Prophet
 - Cambiamos el **nombre de las columnas** “date” y “sales” a “**ds**” y “**y**” respectivamente

```
1 # Preparamos los nombres de columnas para Prophet
2 data_processed.rename(columns={'date':'ds', 'sales':'y'}, inplace=True)
```

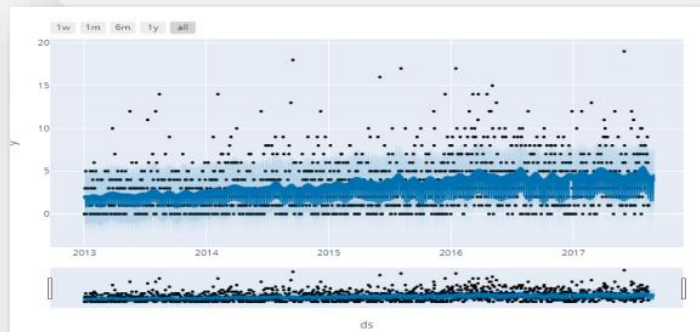
```
1 data_processed
```

	id	ds	store_nbr	family	y	onpromotion	month	day	day_of_week	sales_media_movil_30_dias_lag16	city	state
0	0	2013-01-01	1	6.043813	0.000	0	1	1	1	0.000000	551.338717	548.895202
1	1	2013-01-01	1	0.109488	0.000	0	1	1	1	4.666667	551.338717	548.895202
2	2	2013-01-01	1	3.680752	0.000	0	1	1	1	0.000000	551.338717	548.895202

Entrenar modelo

- ❑ Establecemos los **hiperparámetros** con los que va a trabajar nuestro modelo.
- ❑ Generamos una **primera predicción con un dataset reducido** para ver cómo funciona, y generamos gráficas para ver el resultado.

```
7 prophet_params = {'growth': 'linear',  
8                   'yearly_seasonality': True,  
9                   'weekly_seasonality': True,  
10                  'daily_seasonality': False,  
11                  # 'holidays': festivos,  
12                  'seasonality_mode': 'multiplicative',  
13                  # 'holidays_prior_scale': [5,10,20]  
14 }
```





Entrenamiento y predicción

- Una vez visto que la predicción es coherente entrenamos **un modelo para cada serie temporal** (para cada familia de cada tienda, 1782 series temporales) y solicitamos la predicción de los siguientes 16 días.

```
m.fit(serie)
future = m.make_future_dataframe(periods=16)
# No está prediciendo los días de navidad, así que los quito de los regresores
oil_processed_filtered = oil_processed.drop(pd.date_range(start = '2013-01-01', end = '2017-08-31' ).difference(future.ds).tolist())
oil_processed_filtered = oil_processed_filtered.reset_index()
future['dcoilwtico'] = oil_processed_filtered['dcoilwtico']

forecast = m.predict(future)
forecast_selected_columns = forecast[['ds', 'yhat']].tail(16)
forecast_selected_columns['shop_nbr'] = t
forecast_selected_columns['family'] = f
full_forecast = pd.concat([full_forecast, forecast_selected_columns])
```



Preparar fichero para Kaggle

- ❑ **Ordenamos las filas** por date + store_nbr (como string) + family para obtener el mismo orden que el dataset original y asignamos el ID a cada fila a partir de 3000888

```
[ ] 1 # Convierto store_nbr a string para que lo ordene alfabéticamente como en test.csv (1, 10, 11, 12 ... 54, 6, 7, 8, 9)
    2 full_forecast['store_nbr'] = full_forecast['store_nbr'].astype(str)
    3 full_forecast.sort_values(['ds', 'store_nbr', 'family'], ascending=[True, True, True], inplace=True)
    4 full_forecast['yhat'][full_forecast['yhat'] < 0.01] = 0
    5
    6 submission = pd.DataFrame()
    7 first_id = 3000888
    8 submission.insert(0, 'id', range(first_id, first_id + len(full_forecast)))
    9 submission['sales'] = full_forecast['yhat']
   10 submission.to_csv( os.path.join( rootdir, 'submission.csv'), index=False, encoding='utf-8' )
```

Evaluación

☐ Obtenemos un error bastante **elevado**



submission_base.csv

Complete · 5d ago · Esta predicción es la primera. Solo tiene metido como regresor el precio del petróleo, y no están revisados los valores muy próximos a cero ...

3.54423

✦ Corrección

- ❑ Observamos **valores de ventas negativos en la predicción y otros muy próximos a cero.**

Comparando el dataset y la predicción llegamos a la conclusión de que esos valores tan bajos deberían ser cero. Modificamos el fichero de predicciones y volvemos a subirlo sin ningún éxito.



submission_negativos_zero.csv

Complete · 5d ago · Aquí he convertido los valores negativos y muy bajos (<0.01) a cero

3.54424



Análisis del resultado

- ❑ Evaluamos posibles causas para obtener un error tan elevado.
 - **Fallos al procesar los datos.** El punto más probable por su complejidad es el momento en el que se añaden y codifican los datos de las vacaciones.
 - También hemos podido **fallar a hacer Target Encoding**, calculando la media con los datos equivocados.
 - Fallo en la **elección de hiperparámetros** de Prophet.
 - Fallo al **ordenar el fichero CSV** para Kaggle.



Análisis del resultado

- ❑ Comprobaciones y pruebas:
 - Revisar la transformación y traslado de los datos de un dataframe a otro
 - Cambios en los hiperparámetros
 - Ordenado del fichero de Kaggle

- ❑ Al no tener éxito se decide recomenzar **simplificando**:
 - **Prescindimos de Target Encoding y de Label Encoding**
 - **Eliminamos el nombre de la ciudad y la región.** Nos han servido para obtener los días festivos pero probablemente estén añadiendo ruido
 - **Simplificamos la selección de hiperparámetros**

```
m = Prophet(changepoint_prior_scale=10)
m.fit(serie)
```

✦ Nuevo resultado

❑ **Notable mejoría** pese a haber simplificado todo el proceso.



submission_3.csv

Complete · 4d ago · Siguiendo tutorial y tomando solo datos de los últimos dos años

0.67299

Conclusiones

Conclusiones

- ❑ Es muy **fácil cometer errores en el procesamiento de datos** en un dataset complejo. Simplificar nos ha ayudado a minimizar el riesgo de error humano y obtener mejor resultado.
- ❑ Detectar nuevas features que aporten información relevante es intuitivo. **Detectar las que generan ruido no tanto.**
- ❑ No hemos podido exprimir la metodología al consumir demasiado tiempo en pruebas y errores. **Habría sido necesario:**
 - Hacer un **subset de validación** desde los datos de train y calcular el error en cada prueba.
 - Aplicar **validación cruzada** para encontrar hiperparámetros más óptimos.

```
print(Gracias)
```

```
-----  
NameError Traceback (most recent call last)  
<ipython-input-1-998a4afd91ed> in <cell line: 1>()  
----> 1 print(Gracias)
```

```
NameError: name 'Gracias' is not defined
```