**Prediciendo ingresos tributarios en México**

**Organización de carpeta**

Camilo Arias

Verano 2019

Manual para usar scripts de *Python* del proyecto de predicción de ingresos tributarios. El manual se organiza por tareas que se pueden realizar.

Para predecir nuevas variables, hay que seguir las instrucciones que se indican en: Automatizar descarga de nuevos datos, después importar datos, después, identificar los mejores modelos y finalmente en Predicción, una vez seleccionados los mejores modelos.

Contenido

[Automatizar descarga de nuevos datos: 1](#_Toc18585051)

[INEGI: 1](#_Toc18585052)

[BANXICO: 2](#_Toc18585053)

[FED: 2](#_Toc18585054)

[Datos abiertos (Estadísticas Oportunas): 3](#_Toc18585055)

[Importar datos 3](#_Toc18585056)

[Actualizar Excel de ingresos brutos usando cuadros preliminares del SAT. 4](#_Toc18585057)

[Describir datos 5](#_Toc18585058)

[Predicción, una vez seleccionados los mejores modelos 6](#_Toc18585059)

[Identificar los mejores modelos 8](#_Toc18585060)

# Automatizar descarga de nuevos datos:

Para automatizar la descarga de datos, hay que modificar el script que se conecta con la API de los proveedores. El script puede descargar datos de INEGI, BANXICO y de la FED de San Luis. Además, el script descarga datos del historial de ingresos fiscales de Datos Abiertos.

## INEGI:

1. Ir a <https://www.inegi.org.mx/servicios/api_indicadores.html>
2. Identificar serie que se quiere descargar en la pestaña “constructor de consultas”
3. En el cuadro de la derecha que dice “URL API” identificar la clave del indicador. Son 6 dígitos ubicados después de INDICATOR y antes de *es*: INDICATOR/xxxxxx/ES
4. Abrir script config.py. Añadir al diccionario INEGI el nombre con el que uno quiere llamar a la nueva serie mapeado a la clave. Por ejemplo, si vamos a incluir a la Población Económicamente Activa, añadiríamos: ‘PEA’ : ‘444557’
5. Abrir script download.py. Modificar la función *get\_files\_inegi()* de la siguiente manera:
   1. Incluir un nuevo argumento con el nombre de la serie igualado a False. Por ejemplo, añadir *PEA = False*  a los argumentos de la función.
   2. Añadir un condicional como todos los que están:

if <new\_serie>:

download\_inegi(<new\_serie>)

1. Modificar el Jupyter Notebook *download.ipynb,* incluyendo el nuevo argumento en la función get\_files\_inegi ()

## BANXICO:

1. Ir a <https://www.banxico.org.mx/SieAPIRest/service/v1/;jsessionid=c87172ba5a6fa8cfd219c31f8850>
2. Identificar la serie que se quiere descargar en el catálogo de series. Identificar el idSerie.
3. Abrir script config.py. Añadir al diccionario BANXICO el nombre con el que uno quiere llamar a la nueva serie mapeado a la clave. Por ejemplo, si vamos a incluir a la CAT de Banjército: ‘cat\_banjercito’ : ‘SF324163’
4. Abrir script download.py. Modificar la función *get\_files\_banxico()* de la siguiente manera:
   1. Incluir un nuevo argumento con el nombre de la serie igualado a False. Por ejemplo, añadir cat\_banjercito *= False*  a los argumentos de la función.
   2. Añadir un condicional como todos los que están. Por ejemplo:

*if cat\_banjercito:*

*download\_inegi(‘cat\_banjercito’)*

1. Modificar el Jupyter Notebook *download.ipynb,* incluyendo el nuevo argumento en la función *get\_files\_banxico*()

## FED:

1. Ir a <https://fred.stlouisfed.org/>
2. Identificar la serie que se quiere descargar. Para ello, encontrar la serie e ir a la página de la serie, y ver el id que aparece entre paréntesis al lado del nombre.
3. Abrir script config.py. Añadir al diccionario FED el nombre con el que uno quiere llamar a la nueva serie mapeado a la clave. Por ejemplo, si vamos a incluir a la tasa de desempleo: ‘unemp’ : ‘LNS14000024’
4. Abrir script download.py. Modificar la función *get\_files\_fed()* de la siguiente manera:
   1. Incluir un nuevo argumento con el nombre de la serie igualado a False. Por ejemplo, añadir unemp *= False*  a los argumentos de la función.
   2. Añadir un condicional como todos los que están. Por ejemplo:

*if unemp:*

*download\_fed(‘unemp’)*

1. Modificar el Jupyter Notebook *download.ipynb,* incluyendo el nuevo argumento en la función *get\_files\_fed()*

## Datos abiertos (Estadísticas Oportunas):

Los datos que vemos en Estadísticas Oportunas de la Finanzas Públicas se pueden obtener en archivos CSV que contienen todos los ingresos y gastos del sector público. Estos están en <https://datos.gob.mx/busca/dataset/estadisticas-oportunas-de-finanzas-publicas-principales-indicadores-fiscales>.

La función *download\_fiscal\_data()* descarga los archivos zip y usa los archivos CSV para obtener los ingresos fiscales relevantes. El formato de los archivos csv no es el ideal, por lo que la extracción de los datos de estos archivos se realiza usando en la función *get\_taxes\_from\_csv\_of\_ingresos\_fiscales\_netos()*. Esta función se queda con aquellos impuestos que están en el diccionario INGRESOS\_FISCALES del archive o config. Para añadir una serie de ingresos fiscales:

1. Identificar la clave asociada a la serie. Para esto, descargar el siguiente zip: <https://www.secciones.hacienda.gob.mx/work/models/estadisticas_oportunas/datos_abiertos_eopf/ingreso_gasto_finan_hist.zip> .
2. Abrir el archivo csv y identificar la CLAVE\_DE\_CONCEPTO asociada con el ingreso que quieras descargar.
3. Incluir la clave en el diccionario INGRESOS\_FISCALES y el nombre de la serie.
4. Ahora, cada que se descarguen los ingresos fiscales, se descargarán estos datos también. Volver a correr en el Jupyter notebook *download.ipynb* la función para descargar datos fiscales.

# Importar datos

Es recomendable crear una función que importe los datos y los convierta a *Pandas DataFrames.* Esto ayuda a tener consistencia en el tipo de series, en los nombres y el índice. En el script hay varias funciones que se usan para importar diferentes datos.

1. *load\_calendario\_lif*: Importa el Excel con el calendario de la Ley de Ingresos de la Federación.
2. *load\_ingresos\_fiscales\_brutos*: Importa le Excel con ingresos fiscales brutos.
3. *load\_ingresos\_fiscales\_netos*: Importa el csv de ingresos fiscales netos.
4. *load\_ingresos\_fiscales*: Importa los ingresos fiscales netos y los ingresos fiscales brutos y los carga en un solo DataFrame. Usa las funciones para cargar los ingresos fiscales brutos y netos.
5. *load\_inegi\_indic*: Importa los datos que se descarguen de INEGI con frecuencia mensual que no se deban transformar a reales ni se deba cambiar la base. Estos son todos los IGAES, IMAI, Confianza al consumidor, Indicadores adelantado y coincidente y EMEC.
6. *load\_banxico*: Importa los datos descargados de Banxico.
7. *load\_fed*: Importa los datos descargados de la FED.
8. *load\_inpc*: Importa el INPC y lo convierte a la base del año actual usando el promedio de los valores observados del año actual.
9. *load\_pib\_r*: Importa el PIB descargado de INEGI. Lo convierte a pesos del año actual, y lo convierte a frecuencia mensual si se pasa el argumento *monthly* = True.
10. *load\_balanza\_comercial*: Importa las series de exportaciones e importaciones que se descargan de INEGI, los convierte a pesos usando el tipo de cambio promedio y los convierte a pesos constantes del último año.
11. *extract\_from\_cuadro\_preliminar*: Importa el Excel de cuadro preliminar, creando un DF con los datos de ingresos brutos, netos, compensaciones, devoluciones y regularizaciones.
12. *extract\_from\_cuadro\_isr\_iva\_ieps*: Importa el Excel de IVA, ISR e IEPS los valores de los IEPS desglosados.

Para importar nuevos datos, hay dos opciones: Crear nuevas funciones específicas, o usar alguna de las funciones actuales. Si se trata de un indicador del INEGI, tal vez lo mejor sea usar la función *load\_inegi*(). Si es la serie de ingresos fiscales brutos desde 2000, sería ideal crear una función nueva que lea el archivo, convierta el índice a fecha, los convierta en valores reales y retorne un Pandas DF. Para ello, habrá que se consistentes con la ubicación de los ingresos reales.

# Actualizar Excel de ingresos brutos usando cuadros preliminares del SAT.

Para actualizar los ingresos tributarios brutos y hacer análisis de eficiencia y de gastos fiscales, hay que hacer lo siguiente:

1. Abrir *machote\_eficiencia\_recaudatoria.ipynb*
2. En los parámetros:
   1. indicar True en ‘actualizacion\_recaudacion’
   2. Indicar la ruta al Excel con el cuadro preliminar de recaudación en ‘archivo\_datos\_fiscales\_brutos\_actualizar’
   3. Indicar el mes que se va a actualizar en ‘mes\_a\_actualizar’
3. Correr la sección Actualizamos datos de recaudación del Jupyter notebook. (Revisar los valores que se van a actualizar después de la primera celda de la sección para confirmar que son correctos).
4. Los ingresos brutos actualizados se guardarán en ../inputs/ingresos\_tributarios\_desglosados\_updated.xlsx.
5. Cada que se desee cargar los ingresos tributarios brutos (ya sea usando la función *load\_ingresos\_fiscales()*, o *load\_ingresos\_fiscales\_brutos()*), será necesario indicar en el argumento excel\_brutos la ruta al nuevo archivo. Si se quiere sobrescribir el archivo original para no tener que indicar esto, habrá que reemplazar el archivo *ingresos\_tributarios\_desglosados* con el nuevo archivo.

# Describir datos

Para obtener estadísticas descriptivas podemos usar algunas de las funciones del script descriptive.py y Jupyter notebook. En descriptive.py las funciones más útiles son:

1. *plot\_series()*. Es la función base para graficar series. Toma como argumentos el DF, la lista de variables a graficar, el título, subtítulo, la lista de nombres a usar para la leyenda, la ruta para guardar el gráfico, fecha mínima para graficas (Cuando no se quieren grafiar todos los datos), fecha máxima, la unidad de las divisiones del eje (‘auto’, ‘monthly’, ‘yearly’), la frecuencia (Cada cuantas unidades hacer un tick, por ejemplo, si se selecciona 1 y en ticks se indica ‘monthly’, se colocarán divisiones cada mes), el tamaño de la figura (x, y), una opción para posicional la legenda afuera del área del gráfico, la resolución (100 es buena resolución), una nota al pie y una línea horizontal.
2. *transformation()*. Realizar transformación a una serie.
3. *revert\_transformation().* Revertir una transformación. Las transformaciones de diferencias requieren de valor y fecha inicial.
4. *plot\_acf\_pacf().* Gráfica de auto correlación y auto correlación parcial.
5. *cross\_tab()*. Hacer tabla comparativa de los valores de una serie de un año contra otro año. Por ejemplo, sirve para comparar los ingresos brutos de 2019 con los ingresos de 2018. Se pueden ver los resultados acumulados y en flujo. Se pueden ver las diferencias porcentuales.
6. *cross\_tab\_lif()*. Función específica para comparar la recaudación con la Ley de Ingresos.
7. *seasonally\_adjust()* Función que utiliza R para hacer ajuste estacional con la metodología del Censo de Estados Unidos.

Podemos usar los siguientes Jupyter notebook como referencia:

* *Pipeline\_descriptive.ipynb*
* *Pipeline\_multivariate\_descriptive.ipynb*
* *Machote\_eficiencia\_recaudatoria.ipynb*

# Predicción, una vez seleccionados los mejores modelos

Predecir con modelos econométricos o de Machine Learning se puede hacer usando las funciones que están en el script models\_multivariate.py, tal como se usan en los Jupyter Notebook *Estimaciones\_cierre.ipynb* y *Pipeline\_multivariate\_predict.ipynb.*

Siguiendo el notebook de *Estimaciones\_cierre*, el proceso para predecir los siguientes 18 meses es el siguiente:

1. Importar los datos usando las funciones de ‘load’
2. Definir los parámetros de las estimaciones usando el diccionario params. Ahí indicamos:
   1. La fecha de la primera observación que queremos predecir.
   2. La fecha del último valor que tomaremos como observado. Esto es importante porque normalmente contamos con datos cuya última observación difiere en fecha final. Por ejemplo, podemos tener el tipo de cambio promedio hasta el último mes, el IMAI hasta cierto mes y el IGAE hasta dos meses antes. Para solucionar este problema, construimos una base de datos que tenga todos los valores “observados” hasta esa fecha. Eliminamos aquellas observaciones que pasan esta fecha, y predecimos aquellas otras cuya última observación tiene un rezago de uno o dos periodos. Usamos recursivamente un modelo VAR para esto.
   3. Periodos a predecir: Número de meses.
   4. MODELOS. Cada modelo lo especificamos en un diccionario que indica las variables endógenas, el tipo de modelo, los parámetros, las variables exógenas, etc. Después del análisis de modelos que hicimos, los modelos que seleccionamos para estimar el cierre son los siguientes:
      1. Modelo para predecir el marco macro de Estados Unidos. Tal como esta escrito, es un modelo VAR, con las variables endógenas: tbill\_3meses\_mensual, cons\_price\_index\_us, ind\_prod\_ind\_us, trade\_weighted\_exchange\_rate, commodity\_price\_index, transformadas en log diff, con dummies para 2000 y 2009, con fecha inicial de enero 1992 y con 12 meses como máximo número de lags, criterio de información bayesiano y una constante.
      2. Modelo para predecir marco macro de México. La lectura es similar, pero añadimos variables exógenas que son las que predecimos con el marco macro de USA.
      3. Modelos ARIMAS: Especificamos 6 modelos ARIMA/SARIMA. Cada uno para predecir uno de: Ingresos tributarios totales, ingresos tributarios sin gasolina, ISR, IVA, Impuesto a las importaciones e IEPS.
      4. Modelos VAR: Especificamos dos modelos VAR, uno para las variables endógenas de ISR, IVA, Importaciones e IEPS, y otro para ISR e importaciones netas.
   5. Modelos Machine Learning:
      1. Un modelo de ML para importaciones y otro para IVA.
3. La siguiente etapa es prepara los datos para que todos tengan misma fecha final. Hacemos el *nowcast* de las variables que necesitan, tanto de México como de USA. Este proceso usará los parámetros de fecha final que indicamos. Lo que hace es identificar las variables que no están actualizadas hasta la fecha última indicada, y estima un SARIMA (3, 0, 3)(1, 0, 1)12 para cada una de ellas transformadas en log-diff, usando como variables exógenas a todas las otras variables que sí están hasta la ultima fecha. Este procedimiento no es el más riguroso, ni el más formal, pero sirve para obtener una aproximación decente.
4. Predecimos el marco macro de USA. Usará los parámetros que indicamos. Usamos el modelo VAR que describimos en los parámetros. Guardamos la predicción en el objeto *prediction\_us*.
5. Predecimos el marco macro de México. Usamos el modelo VAR que indicamos en los parámetros. Guardamos la predicción en *prediction\_mx*. Para ver el marco macro que estimamos, convertimos el DataFrame de *prediction\_mx* a valores anuales.
6. Predecimos los ingresos tributarios. Estimamos los modelos ARIMA, VAR y de Machine Learning.
7. Vemos las predicciones y obtenemos gráficas.

# Identificar los mejores modelos

Para realizar el ejercicio completo de predicción nos enfocaremos en el Jupyter notebook *Pipeline\_multivariate\_predict*. En este notebook estimamos cientos de modelos para cada una de las variables de interés, obtenemos la precisión de cada uno de los modelos, y exportamos los datos de precisión a una serie de archivos CSV que analizamos en el Jupyter Notebook *Pipeline\_analize.*

Un componente esencial de este procedimiento es indicar las especificaciones que vamos a probar para cada uno de los modelos. Es decir, necesitamos indicar los parámetros de los SARIMAS, ARIMAS, VAR, DT, GB, y RF que vamos a estimar. El espectro es muy amplio, desde únicamente estimar una especificación para cada uno de estos modelos, hasta estimar todas las combinaciones posibles. Para ello, usamos el script *grid* como auxiliar. En *grid,* creamos un diccionario de modelos econométricos y de modelos de machine learning, y los agrupamos en tres grupos: *small, médium y large*. *Small* es un grupo reducido de modelos, sirve para probar el funcionamiento del notebook. *Medium* tiene un poco mas de modelos y *large tienen* muchos modelos. El procedimiento es el siguiente:

1. Importar los datos
2. Especificar los parámetros de la predicción. Esta es la clave del ejercicio. En los parámetros se indica cuales son los modelos que vamos a probar:
   1. Beginnings: Cuales van a ser los cortes de predicción. En este caso, indicamos cortes para Julio de 2015, 2016, 2017 y 2018.
   2. Periods: Cuantos meses vamos a predecir.
   3. econometric\_models: Este parámetro indica cuales serán los modelos econométricos que vamos. Indicar “small” , “médium” o “large”,
   4. ml\_models: Este parámetro indica cuales serán los modelos econométricos que vamos. Indicar “small” , “médium”, “large”, o “custom” que es una lista personalizada que creamos en el script *grid.*
   5. Especificar el VAR con el que estimaremos el marco macro externo. Indicamos las variables endógenas, la transformación de las variables endógenas, las dummies exógenas, y la fecha de inicio.
   6. Especificar VAR con el que estimaremos el marco macro de México. En este caso, estimamos tres modelos que se diferencian en las variables endógenas que incluyen.
   7. Especificamos los modelos VAR de ingresos fiscales. Cada uno de estos modelos será estimado con cada uno de los parámetros que especificamos en script *grid.* El primero, por ejemplo, es un modelo VAR con dos variables endógenas: ISR e IVA neto. El parámetro ‘other exog’ sirve para indicar que queremos agregar un DataFrame extra a las variables exógenas. En este caso, es el DataFrame semana\_santa\_y\_tasas, que tiene una serie con dummies para los meses en los que cayó Semana Santa y dos series con las tasas de IVA y de ISR.
   8. Especificamos los modelos ARIMA de ingresos fiscales.
   9. Especificar los modelos de Machine learning de Ingresos fiscales.
3. Construimos el DataFrame conjunto y hacemos el *nowcast.* Aunque no es estrictamente necesario para este ejercicio, pues nunca usaremos los datos mas actuales para evaluar los modelos.
4. Predecimos el marco macro de Estados Unidos. Guardamos las predicciones en un diccionario que mapea fecha de predicción con predicciones, y en una lista que contiene las precisiones de cada una de las predicciones. Estos son prediction\_us y accuracies\_us. Guardamos las precisiones en un csv.
5. Analizamos las precisiones y seleccionamos el mejor parámetro. Después usamos este parámetro para crear el diccionario best\_prediction\_us, que usaremos para las predicciones del marco macro mexicano.
6. Estimamos los modelos VAR de la economía mexicana. Son tres modelos, dos de ellos con variables tributarias. El marco macro lo obtenemos del modelo sin variables tributarias. De cada modelo, obtenemos un diccionario con las predicciones, una lista con las precisiones, y un diccionario comparando los datos anuales que cada modelo arrojo con los datos anuales observados. Para el primer modelo, estos son: *prediction\_mx\_w\_trib, accuracies\_mx\_w\_trib,* y *y\_mx\_diffs\_w\_trib*. Cada una de las listas de precisiones la exportamos a un CSV.
7. Estimamos los modelos VAR de ingresos tributarios. Las precisiones las guardamos en results\_var, que exportamos a un CSV.
8. Estimamos los modelos ARIMA y SARIMA. Guardamos las precisiones en *results\_arima* y las exportamos a un CSV.
9. Estimamos modelos de Machine Learning, guardamos las precisiones en *results\_ml* y lo exportamos a un CSV.

Una vez que contamos con los resultados, los analizamos en el Jupyter Notebook *Pipeline Analize.* En este Jupyter notebook, importamos todos los CSV que creamos, y podemos analizar los resultados según el parámetro que indicamos como el objetivo, que puede ser *rmse\_first6, rmse\_last12, rmse\_first18, mape\_first6, mape\_first18* o *mape\_last12.*