

Reporte de propuestas de  
mejora para el proyecto

# **Sistema Inteligente para pronósticos, basado en series de tiempo**

---

Basado en Objetivos

Por: Br. Jorge Israel Celis Hernández

última edición: 28 de febrero del 2025

## índice

<b>Antecedentes.....</b>	<b>2</b>
<b>1. Jerarquía de Carpetas.....</b>	<b>3</b>
<b>2. Guardado del modelo entrenado.....</b>	<b>5</b>
<b>3. Cargar el modelo y hacer predicciones.....</b>	<b>6</b>
<b>4. Guardado de las gráficas obtenidas en predicción y en entrenamiento.....</b>	<b>8</b>
<b>5. Guardado de la metadata y características de la creación y entrenamiento del modelo:.....</b>	<b>10</b>
<b>6. Identificación de la estabilidad del modelo ante diferentes escenarios.....</b>	<b>12</b>

## Antecedentes

El proyecto que se abordará, en sí, es parte de una de las propuestas de la empresa consultora y proveedora de soluciones TI: **GESTIÓN DE INNOVACIÓN EN TECNOLOGÍA INFORMÁTICA S.C.P. “Grupo Consultores**, la cual ofrece productos y servicios hacia clientes quienes buscan la actualización, migración y/o actualizaciones de sus sistemas. En particular, este proyecto cuenta con una arquitectura y una infraestructura prediseñada, y que ha sido catalogada como un prototipo por parte de la empresa, el proyecto ha funcionado, pero con las siguientes limitaciones que se pudieron identificar:

- Se tiene en forma de un solo script en un notebook de Jupyter.
- Se extraen los datos por medio de PyODBC, una biblioteca de Python.
- Es un programa de consola
- Está programado en Python.
- Usa la biblioteca de Keras para el entrenamiento del modelo
- Usa librería de Python Matplotlib para representar las predicciones por medio de una gráfica.
- El sistema, no puede volver atrás, y no guarda lo aprendido por el modelo, sino que, tiene que volver a entrenarse el modelo para luego volver a analizar los datos.
- No presenta una información adecuada que indique los problemas en específicos que se plantean resolver.
- No hay documentación del código o algo que indique la funcionalidad de los bloques de código
- Si bien la ideología y los objetivos iniciales resultaron efectivos. Tienen aspectos y puntos que pudieran tener una mejoría, que beneficie al usuario final del sistema.

En resumen, el proyecto se presenta de la siguiente manera: permite hacer importaciones para tratamiento de datos por medio de PyODBC, pero solamente con SQL Server, el modelo está entrenado para funcionar de una forma nada más, una vez el modelo entrenado, concluye con los datos y la gráfica modo de resumen final. El sistema fue escrito en Python, y con librerías orientadas al desarrollo de redes neuronales para Inteligencia Artificial. Lo que se buscaba en esta versión era que se pudiera hacer un análisis de datos.

A continuación se presentan una serie de mejoras hacia el sistema, cuales a medida de la marcha del desarrollo y evolución, serán sugeridas hacia los responsables de la administración del proyecto.

## 1. Jerarquía de Carpetas

La jerarquía de carpetas es un aspecto fundamental en el desarrollo del proyecto, ya que, representa donde se guardará no solamente la información contenida del proyecto, sino también el entrenamiento del modelo, y como una casa, las direcciones desde donde la información será guardada y recuperada. La punto principal para la jerarquía de carpetas es la siguiente:













- Una carpeta donde se contendrán los modelos.
- Se sugiere que los modelos deben estar identificados por la fecha en la que fueron entrenados, si se hacen entrenamiento de varios modelos, podrían ser identificados por el día en el que fueron entrenados.
- Si hay más de dos modelos entrenados el mismo día, estos, tendrán dos carpetas distintas, siendo que, su identificación es la hora en la que fueron entrenados.
- Cada carpeta de modelos entrenados contendrá dos carpetas más: una que contemple los datos arrojados o gráficas obtenidas cuando este fue entrenado, así como otra que arroje los resultados de las predicciones realizadas
- En el caso de cuando se hace el entrenamiento, será: **trainedModel\_dataPredict**, que indicará que contiene las gráficas o los datos obtenidos cuando el modelo pase por entrenamiento
- En el caso de cuando se hacen predicciones, será: **reonstruredModel\_dataPredict**, que hace referencia a cuando el modelo es recuperado y se realizan predicciones.
- Dentro de la carpeta **reonstruredModel\_dataPredict**, se propone una nomenclatura similar a la de los modelos, siendo que, por cada vez que se pase por una predicción de acuerdo a ese mismo modelo, se creará una carpeta con los datos de la fecha y la hora con la que fueron obtenidos.

En resumen:

```

PS C:\GitHub\pronostico\pronostico\models> tree
Listado de rutas de carpetas
El número de serie del volumen es 04DB-70D2
C:..
├── 2025-02-21
│   └── model-training2025-02-21_22-19-13
│       └── trainedModel_dataPredict
│           └── 2025-02-21_22-19-13
├── 2025-02-24
│   ├── model-training2025-02-24_13-41-57
│   │   ├── reconstruredModel_dataPredict
│   │   │   └── 2025-02-24_13-42-37
│   │   └── trainedModel_dataPredict
│   │       └── 2025-02-24_13-41-57
│   └── model-training2025-02-24_15-50-15
│       ├── reconstruredModel_dataPredict
│       │   ├── 2025-02-24_15-50-39
│       │   └── 2025-02-24_15-52-59
│       └── trainedModel_dataPredict
│           └── 2025-02-24_15-50-15
└── 2025-02-25
    ├── model-training2025-02-25_15-41-40
    │   ├── reconstruredModel_dataPredict
    │   │   └── 2025-02-25_15-43-25
    │   └── trainedModel_dataPredict
    │       └── 2025-02-25_15-41-40
    └──
PS C:\GitHub\pronostico\pronostico\models>

```

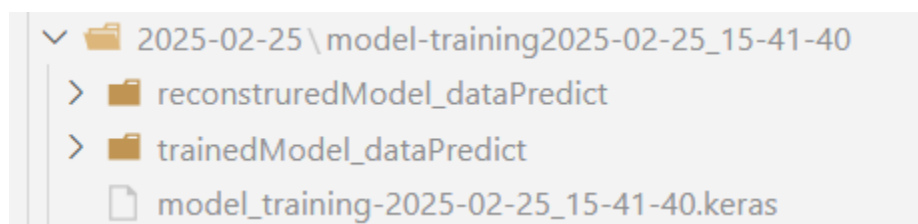
- ▼  models
  -  2025-02-19
  -  2025-02-20
  - >  2025-02-21
  - ▼  2025-02-24
    - ▼  model-training2025-02-24\_13-41-57
      - ▼  reconstruredModel\_dataPredict
        -  2025-02-24\_13-42-37
      - ▼  trainedModel\_dataPredict
        -  2025-02-24\_13-41-57
    - >  model-training2025-02-24\_15-50-15
    - >  2025-02-25

## 2. Guardado del modelo entrenado

De acuerdo a las secuencias y la estructura del flujo de información en la que se prepara la información, desde que es importada, hasta ser predecida, una parte fundamental que se da cuando el modelo ha sido entrenado, es que este no se es guardado, lo cual es ineficiente, puesto que, por cada predicción, el modelo debe pasar por entrenamiento. Para este caso, se aplica un método o un código que guarde el modelo una vez entrenado. La forma para guardar el modelo hasta ahora, se propone una nomenclatura basada en la fecha y la hora en la que fue entrenada.

```
# Parte nueva para guardar los modelos
datetim_e = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
model_path = dirmodels_name+"/model-training"+datetim_e
os.makedirs(model_path, exist_ok=True)
model_name = model_path+'/model_training-'+datetim_e+'.keras'
model.save(model_name)
```

Quedando así:



### 3. Cargar el modelo y hacer predicciones

la parte complementaria de poder guardar un modelo y luego recuperarlo para hacer predicciones es el cargado del modelo, actualmente, la biblioteca de Keras, presenta una api para poder hacer esta acción, la cual toma la información de la ubicación del archivo keras, para luego hacer una predicción de los nuevos datos.

Función obtener módulo:

```
#Reconstruir el modelo
def reconstrued_modelFunc(self,model_path):
    model = load_model(model_path)
    return model
```

Código para hacer las predicciones, que recibe una nueva data o que se anticipa para recibir la nueva consulta con la que hará la predicción

```
def prepareData(self, sqlServerConfig, query, pasos, reconstrued_model):
    with self.get_sqlconnection(sqlServerConfig) as cursor:
        dataPrepare = pd.read_sql_query(query, cursor)
        dataPrepare = self.set_index_datetime(dataPrepare)
        last_day = datetime.strptime(dataPrepare.index.max(), '%Y-%m') + relativedelta(months=1)
        future_days = [last_day + relativedelta(months=i) for i in range(pasos)]
        for i in range(len(future_days)):
            future_days[i] = str(future_days[i])[:10]
        # print("pasaste por aqui")
        future_data = pd.DataFrame(future_days)
        #renombramiento del campo
        future_data.columns = ['date']
        for column in dataPrepare.columns:
            new_data = dataPrepare.filter([column])
            new_data.set_index(dataPrepare.index, inplace=True)
            new_data = self.eliminar_anomalias(new_data)
            x_train, y_train, x_val, y_val, scaler, values = self.create_x_y_train(new_data)
            # x_test = self.reorderData(scaler, values,new_data,pasos)
            reconstrued_model, x_test = self.entrenar_modelo(x_train,y_train,x_val,y_val,scaler,values,new_data,reconstrued_model)
            results = []
            for i in range(pasos):
                parcial = reconstrued_model.predict(x_test)
                results.append(parcial[0])
                x_test = self.agregarNuevoValor(x_test,parcial[0])
            adimen = [x for x in results]
            inverted = scaler.inverse_transform(adimen)
            y_pred = pd.DataFrame(inverted.astype(int))
            future_data[column]= inverted.astype(int)
        future_data = self.set_index_datetime(future_data)

        dataPrepare.index = pd.to_datetime(dataPrepare.index)
        future_data.index = pd.to_datetime(future_data.index)
        return dataPrepare, future_data
```

Codigo completo:

```
#Reconstruir el modelo
def reconstrued_modelFunc(self,model_path):
    model = load_model(model_path)
    return model

#funcion principal para tomar un modelo y hacer predicciones
def modelPredicFuncion(self,sqlServerConfig, query, pasos, model_path):
    reconstrued_model = self.reconstrued_modelFunc(model_path)
    future_date, future_data = self.prepareData(sqlServerConfig,query,pasos,reconstrued_model)
    self.GraphicDataCreate(future_date, future_data, model_path)

def prepareData(self, sqlServerConfig, query, pasos, reconstrued_model):
    with self.get_sqlconnection(sqlServerConfig) as cursor:
        dataPrepare = pd.read_sql_query(query, cursor)
        dataPrepare = self.set_index_datetime(dataPrepare)
        last_day = datetime.strptime(dataPrepare.index.max(), '%Y-%m') + relativedelta(months=1)
        future_days = [last_day + relativedelta(months=i) for i in range(pasos)]
        for i in range(len(future_days)):
            future_days[i] = str(future_days[i])[:10]
        # print("pasaste por aqui")
        future_data = pd.DataFrame(future_days)
        #renombramiento del campo
        future_data.columns = ['date']
        for column in dataPrepare.columns:
            new_data = dataPrepare.filter([column])
            new_data.set_index(dataPrepare.index, inplace=True)
            new_data = self.eliminar_anomalias(new_data)
            x_train, y_train, x_val, y_val, scaler, values = self.create_x_y_train(new_data)
            # x_test = self.reorderData(scaler, values,new_data,pasos)
            reconstrued_model, x_test = self.entrenar_modelo(x_train,y_train,x_val,y_val,scaler,values,new_data,reconstrued_model)
            results = []
            for i in range(pasos):
                parcial = reconstrued_model.predict(x_test)
                results.append(parcial[0])
                x_test = self.agregarNuevoValor(x_test,parcial[0])
            adimen = [x for x in results]
            inverted = scaler.inverse_transform(adimen)
            y_pred = pd.DataFrame(inverted.astype(int))
            future_data[column]= inverted.astype(int)
        future_data = self.set_index_datetime(future_data)

    dataPrepare.index = pd.to_datetime(dataPrepare.index)
    future_data.index = pd.to_datetime(future_data.index)
    return dataPrepare, future_data
```



## 4. Guardado de las gráficas obtenidas en predicción y en entrenamiento

Guardar todos los datos o más bien, guardar los datos predecidos, es una tarea fundamental en cuanto a las necesidades del sistema, sin embargo, es requerido de igual manera, hacer las comparaciones entre los datos obtenidos del sistema, a los que se han predecido. Es por ello, que en este objetivo, se pretende el guardado de cada uno de las gráficas obtenidas, esto se logra, a través del siguiente código:

- Cuando el modelo se ha entrenado, guarda las gráficas en la carpeta **trainedModel\_dataPredict**:

```
#Creamos un directorio para guardar los datos del primer entrenamiento
path = model_path+"/trainedModel_dataPredict/"+datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
os.makedirs(path)

#Configuracion de las imagenes
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('fast')

#Graficar los dataframes
for i in range(len(datos.columns)):
    data = datos[datos.columns[i]][:]
    #Para asignar los valores de los años a los que está sujeto el proyecto se debe guardar en una
    #variable global y luego
    plt.plot(data.index, data, label='Historial 2015 - 2020')
    plt.plot(future_data.index, future_data[future_data.columns[i]], label='Predicción 2021 - 2022')
    xtics = data.index.union(future_data.index)[::8]

    plt.xticks(xtics)
    plt.xlabel('Fecha')
    plt.ylabel('Ventas')
    plt.title('Predicción de la demanda del {p0} para el año del 2021'.format(p0=datos.columns[i]))
    plt.legend()
    plt.figtext(0.01, 0.01, "Realizado el: "+datetime.now().strftime('%H:%M:%S %d-%m-%Y'), fontsize=10, color="gray")
    plt.figtext(0.60, 0.01, "Gestión de Innovación en Tecnología Informática S.C.P | Grupo Consultores*", fontsize=10, color="gray")
    name = path+"/GraphicalPrediction_on_"+str(datos.columns[i])+".jpg"
    plt.savefig(name, dpi=300)
    plt.close() # Cerrar la figura para liberar memoria
```

- Cuando el modelo hace predicciones, se creará una carpeta con una nomenclatura compuesta por la fecha y hora en la que fue hecha. Todo esto dentro de la carpeta **reconstruedModel\_dataPredict**.

```
#Modulo para guardar las gráficas, cuando se hace una predicción
def GraphicDataCreate(self,datos, futureData, model_path):

    #tomamos la ruta del modelo y eliminamos el nombre del modelo
    path = os.path.dirname(model_path)

    #ubicamos la carpeta de predicciones de modelos reconstruidos
    path = path+'/reconstruredModel_dataPredict'

    #Creamos una carpeta de dato entrenados con el modelo reconstruido
    if not os.path.exists(path):
        os.makedirs(path)

    #en el mismo directorio, creamos una carpeta
    path = path+'/'+datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
    os.makedirs(path)

    #Graficar los dataframes
    #considerar almacenar la variable de la columna, ya que, el nombre de la misma puede cambiar
    #Configuracion de las imagenes
    plt.rcParams['figure.figsize' ] = (16, 9)
    plt.style.use('fast')
```




## 5. Guardado de la metadata y características de la creación y entrenamiento del modelo:

Es muy importante obtener los datos y predicciones en el menor tiempo posible, siempre que la tecnología y los recursos lo permitan, sin embargo, al tratarse de información muy privilegiada, y que será indispensable para la toma de decisiones, se requiere que cuando se hagan predicciones el modelo esté bajo las mismas condiciones y configuraciones, con las que se pueda garantizar una mayor veracidad y fidelidad de los datos obtenidos, así como el mismo comportamiento del modelo.

Guardar la metadata del modelo es posible por medio de un archivo json:

```
#Preparamos un archivo para guardar la metadata
metaData = model_path+'/metadata.json'
date, hour = datetime.split('_')
with open(metaData, 'w') as file:
    json.dump([
        {
            '-> FECHA_ENTRENAMIENTO:': date,
            '-> HORA_ENTRENAMIENTO:': hour,
            '-> TOTAL_DATOS:': str(datos.size)
        }
    ], file)
```

Al final, se obtendrá un archivo con la información del modelo:

Nombre	Fecha de modificación	Tipo	Tamaño
 trainedModel_dataPredict	28/02/2025 05:55 p. m.	Carpeta de archivos	
 metadata.json	28/02/2025 05:55 p. m.	Archivo JSON	1 KB
 model_training-2025-02-28_17-55-42.ker...	28/02/2025 05:55 p. m.	Archivo KERAS	25 KB

### Contenido del archivo:

```
[{"-> FECHA_ENTRENAMIENTO": "2025-02-28", "->  
HORA_ENTRENAMIENTO": "17-55-42", "-> TOTAL_DATOS":  
"72"}]
```

Ln 1, Col 1    106 caracteres.    100%    Windows (CRLF)    UTF-8

A medida de que el modelo requiere de más parámetros, se podrán ir guardando en este archivo.

## 6. Identificación de la estabilidad del modelo ante diferentes escenarios

El sistema requiere de todos los posibles escenarios en los que el modelo podría hacer predicciones o desempeñarse es un aspecto fundamental, así que, se propone que el modelo se ponga a prueba y se identifique los puntos claves y se monitoree su estabilidad, al predecir y al hacer un entrenamiento, esto se logra en conjunto al objetivo 5 y monitoreando el nivel de entrenamiento... Actualmente se encuentra trabajando en esta característica.