ANÁLISIS DE DATOS Y VISUALIZACIÓN

PREDICCIÓN DE LOS COSTOS DE SEGUROS MÉDICOS

Integrantes:

- María Juarez
- Fabricio Paredes
- Roberto Valladolid
- Bryan Portilla

UTPL

La Universidad Católica de Loja

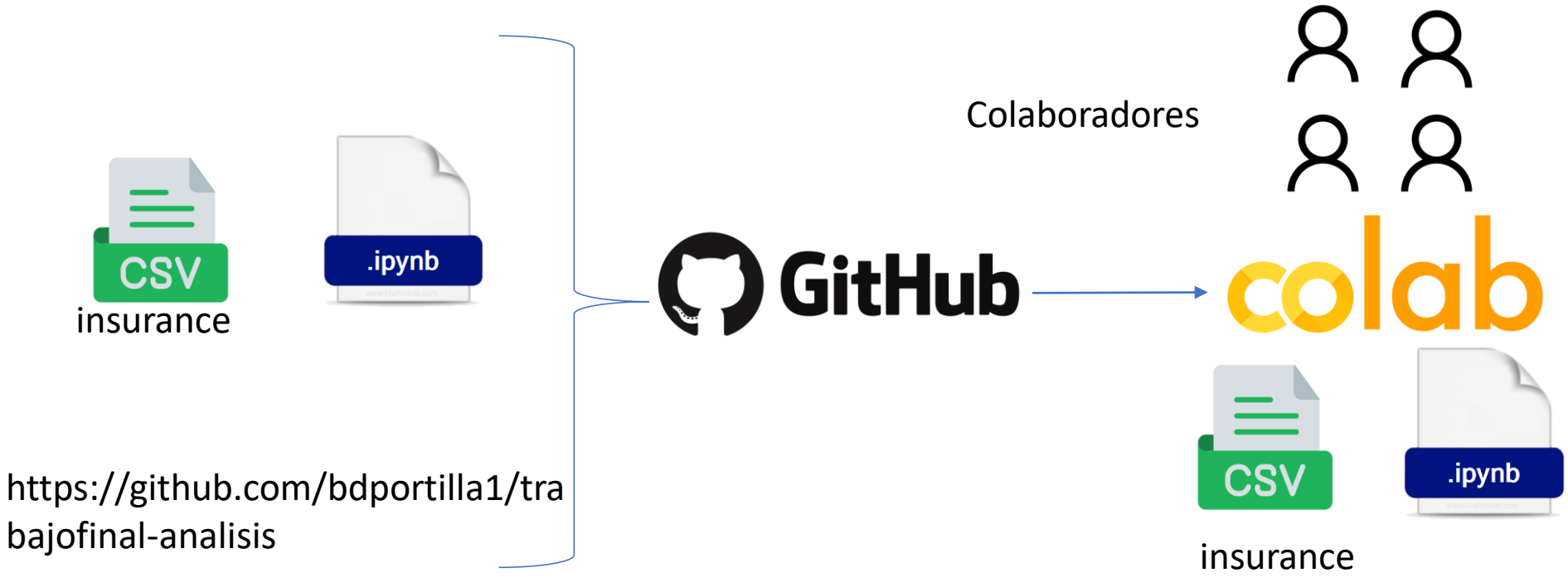
MAESTRIA EN INTELIGENCIA ARTIFICIAL APLICADA

ACTIVIDADES

1. EDA: Explicar el dataset usado, como tamaño, variables, distribuciones y destacar los patrones relevantes encontrados en los datos.
2. Preprocesamiento y preparación de datos: Explicar las tareas de limpieza y transformación de datos realizadas.
3. Modelado: Construcción de modelos, explicación y justificación de métodos usados en la predicción.
4. Evaluación de modelos: Sustentar con el uso de métricas.
5. Explicación e interpretación de resultados

Estructura solución

Fuente del Dataset: KAGGLE
https://www.kaggle.com/datasets/muhammadanwaar101/healthcare-insurance-charges-dataset
Este dataset proporciona información sobre los cargos del seguro médico de las personas en estos registros; incluye varios factores que podrían influir en los cargos del seguro, como la edad, el sexo, el índice de masa corporal (IMC), el número de hijos, el tabaquismo, la región y los cargos del seguro correspondientes.



INFORMACIÓN DEL DATASET

CARACTERISTICAS DEL DATASET

- **Fuente**: Kaggle
- **Nombre**: Insurance.csv
- **Ámbito:** Salud
- **Tamaño:** 1407 filas y 7 columnas
- **Detalle de columnas:**
  - **AGE:** Edad (Años).
  - **Gender:** Género (Male / Female).
  - **Body_Mass_Index (BMI)**: Índice de masa corporal.
  - **Number_of_Children:** Número de hijos que tiene la persona asegurada.
  - **Smoking_Status:** Fumador (yes/no).
  - **Region:** Región donde vive la persona asegurada.
  - **Insurance_Charges:** Cargos del seguro médico (Valor a predecir)

VISIÓN GENERAL DE DATASET

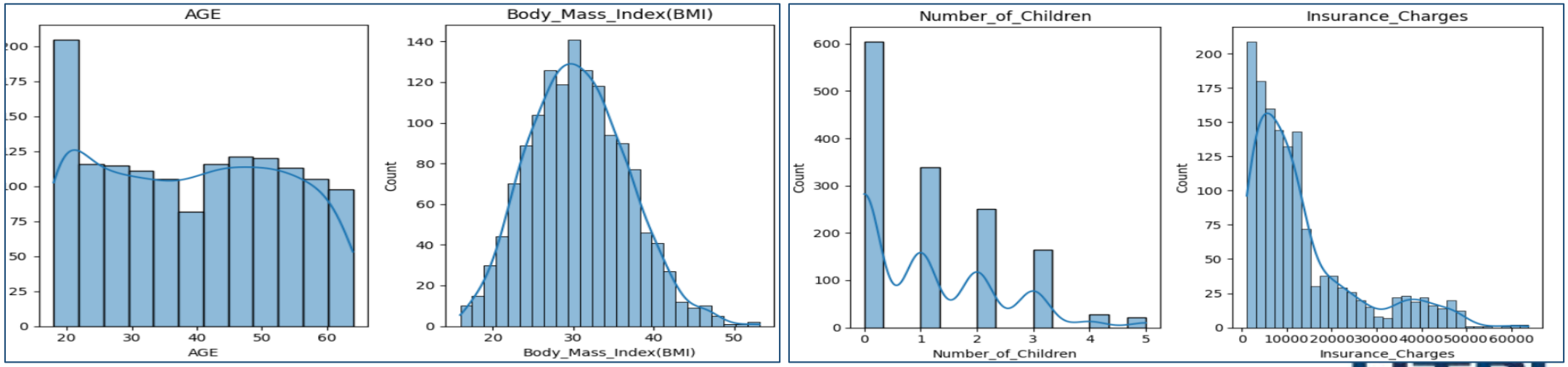
```
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   AGE                   1407 non-null   int64
 1   Gender                1407 non-null   object
 2   Body_Mass_Index(BMI)  1407 non-null   float64
 3   Number_of_Children    1407 non-null   int64
 4   Smoking_Status        1407 non-null   object
 5   Region                1407 non-null   object
 6   Insurance_Charges     1407 non-null   float64
dtypes: float64(2), int64(2), object(3)
```

ANÁLISIS EXPLORATORIO DE DATOS (EDA)

1. Distribución de variables numéricas con KDE
2. Distribución de variables categóricas
3. Distribución de los datos
4. Matriz de correlación
5. Relación entre variables independientes y la variable dependiente.

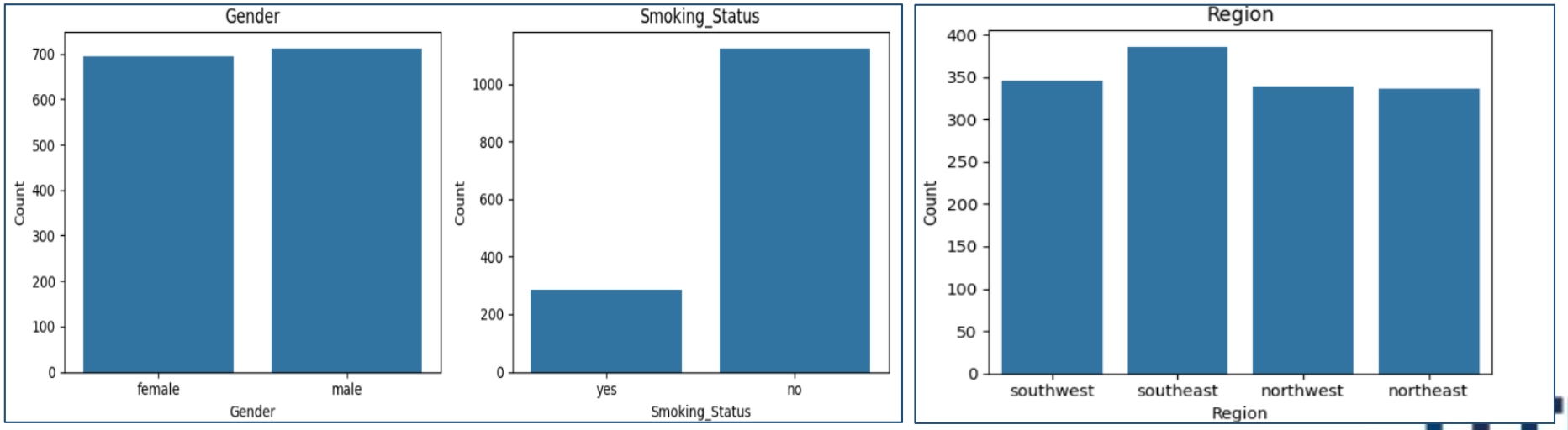
EDA: DISTRIBUCIÓN DE VARIABLES NUMÉRICAS CON KDE

```
int_vars = data.select_dtypes(include = ['int','float'])
num_vars = int_vars.shape[1]
num_rows = math.ceil(num_vars / 4)
print("Distribución de variables numéricas:")

fig, axs = plt.subplots(nrows=num_rows, ncols=4, figsize=(15,(num_rows*4)+1))
axs = axs.flatten()

for i, var in enumerate (int_vars):
  sns.histplot(x=var,data=data,ax=axs[i], kde=True)
  axs[i].set_title(var)
# Si hay más subplots que columnas, ocultar los subplots sobrantes
for j in range(i + 1, len(axs)):
  fig.delaxes(axs[j])

plt.tight_layout()
plt.show()
```



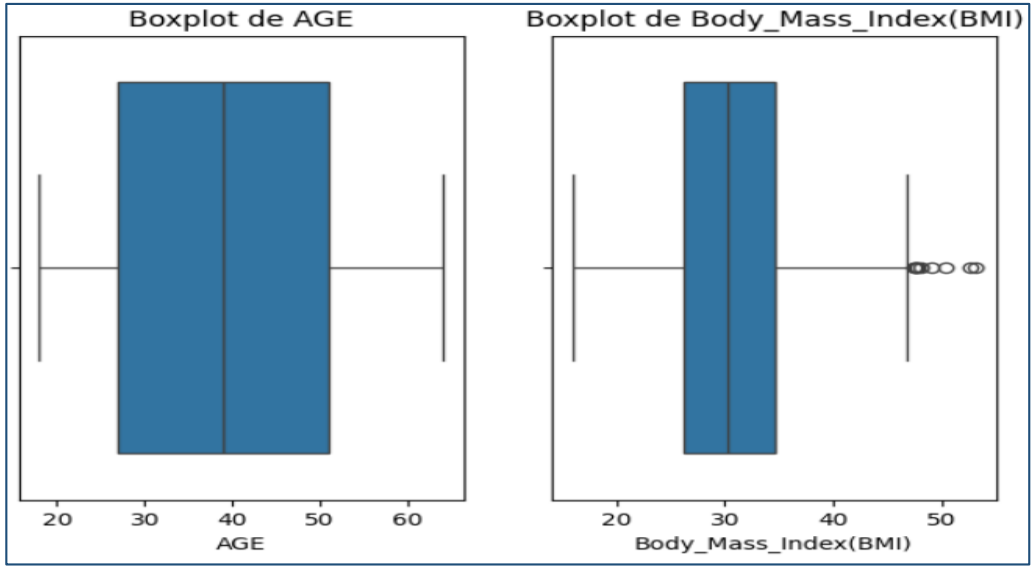
EDA: DISTRIBUCIÓN DE VARIABLES CATEGÓRICAS

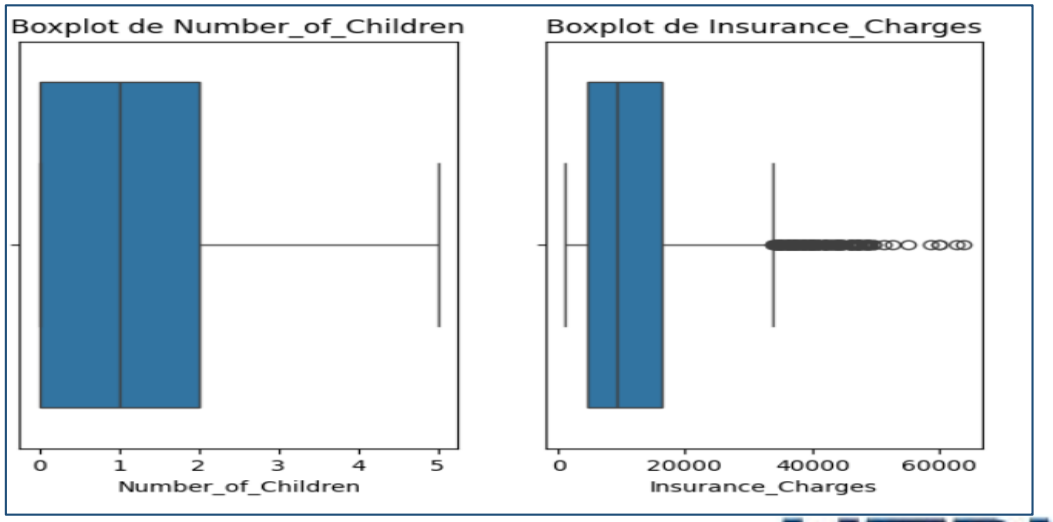
```
cat_vars = data.select_dtypes(include=['object'])
num_varsc = cat_vars.shape[1]
num_rowsc = math.ceil(num_varsc / 3)
print("Distribución de variables categóricas:")

fig, axs = plt.subplots(nrows=num_rowsc, ncols=3, figsize=(15, (num_rowsc*3)+1))
axs = axs.flatten()

for i, var in enumerate(cat_vars):
    sns.countplot(x=var, data=data, ax=axs[i])
    axs[i].set_title(var)
    axs[i].set_ylabel('Count')
# Si hay más subplots que columnas, ocultar los subplots sobrantes
for j in range(i + 1, len(axs)):
    fig.delaxes(axs[j])

plt.tight_layout()
plt.show()
```



EDA: DISTRIBUCIÓN DE LOS DATOS

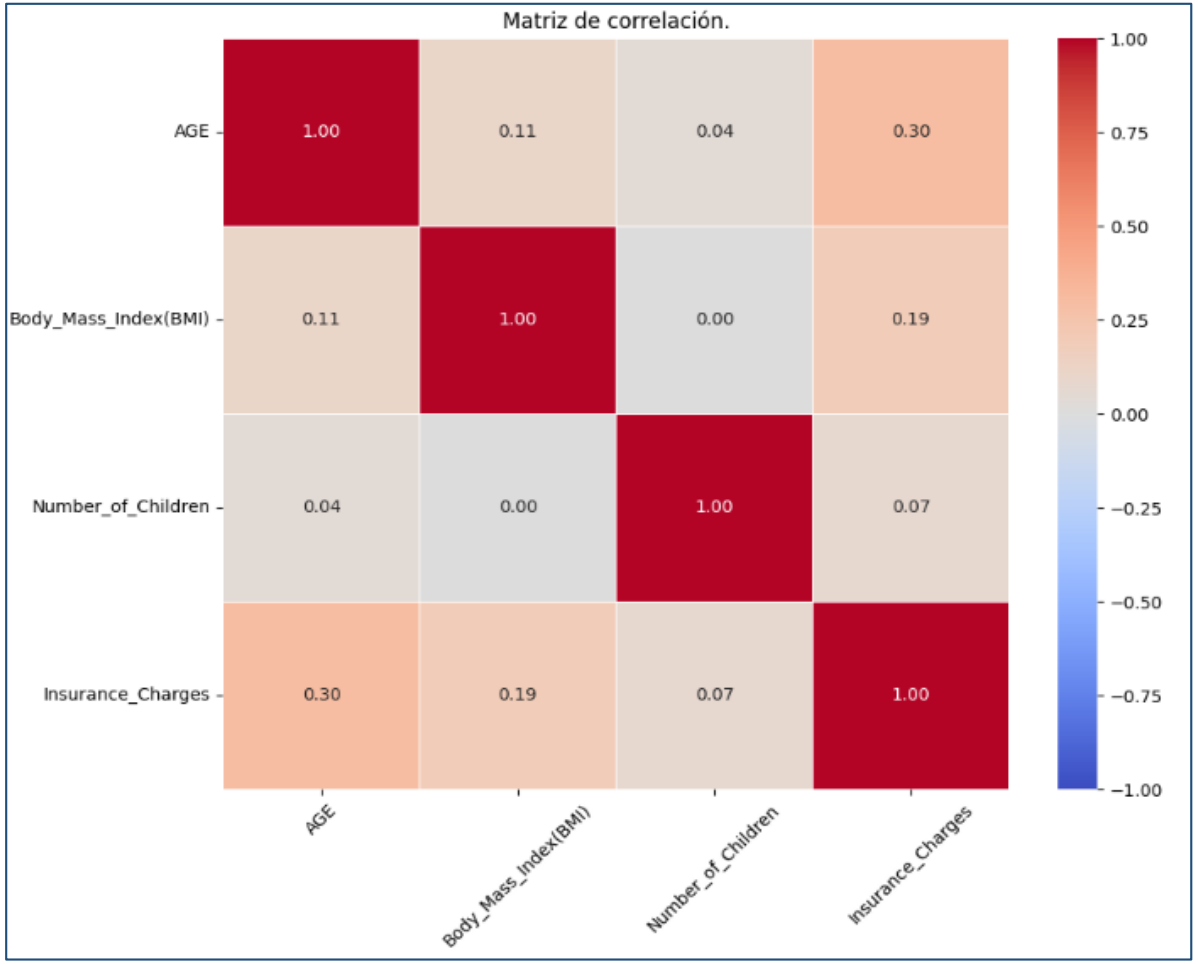
```python
# Boxplot de variables numéricas
print("Boxplot de variables numéricas:")

fig, axs = plt.subplots(nrows=num_rows, ncols=4, figsize=(15,(num_rows*4)+1))
axs = axs.flatten()

for i, columna in enumerate(int_vars.columns):
  sns.boxplot(data=int_vars[columna], orient="h", ax=axs[i])
  axs[i].set_title(f'Boxplot de {columna}')
# Si hay más subplots que columnas, ocultar los subplots sobrantes
for j in range(i + 1, len(axs)):
  fig.delaxes(axs[j])

plt.show()
```





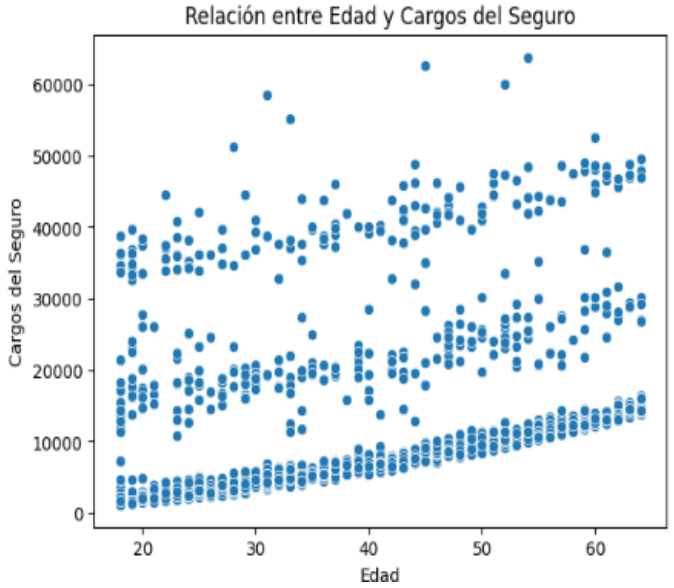
EDA: MATRIZ DE CORRELACIÓN

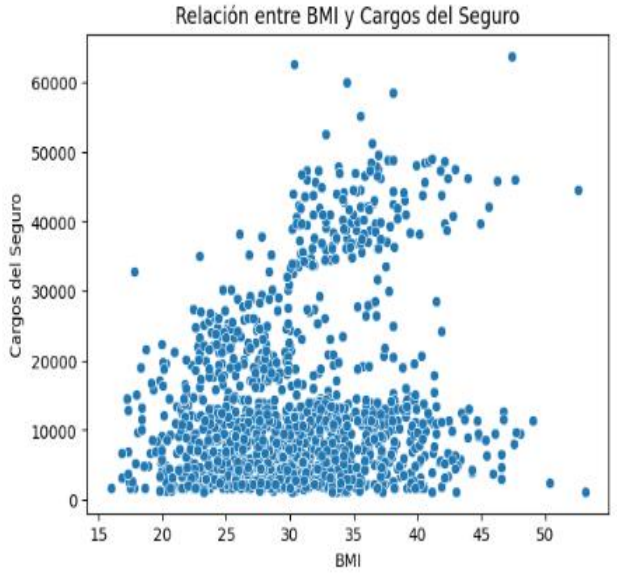
```python
# Filtrar las variables numéricas excluyendo 'Insurance_Charges'
numeric_columns = data.select_dtypes(include=['float64', 'int64'])

# Calcular la matriz de correlación
correlation_matrix = numeric_columns.corr()

# Crear una figura y un eje para la gráfica
plt.figure(figsize=(10, 8))

# Visualizar la matriz de correlación como un mapa de calor con anotaciones
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1, linewidths=0.5)
plt.title('Matriz de correlación.')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```

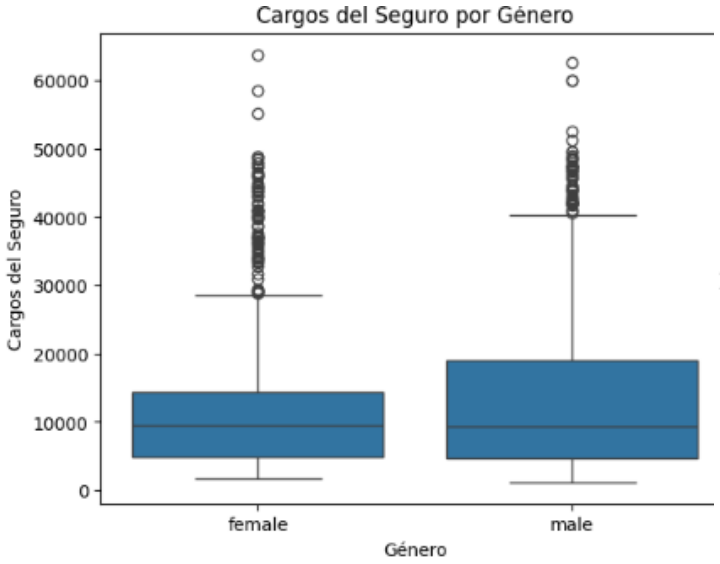


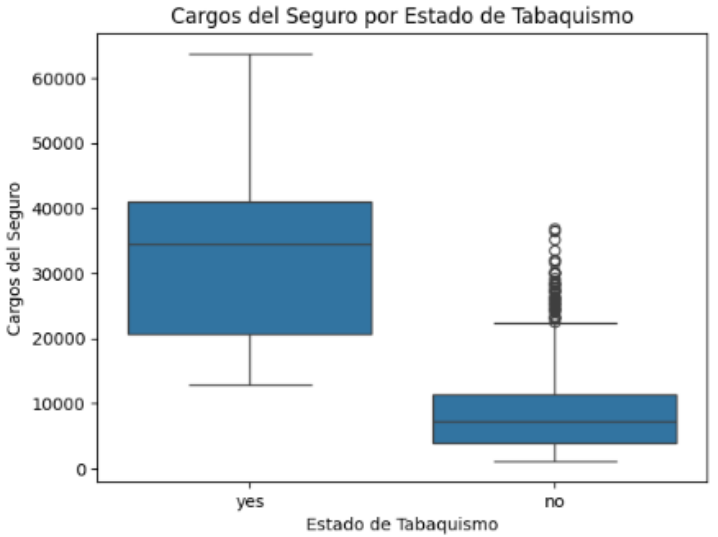
EDA: RELACIÓN ENTRE VARIABLES INDEPENDIENTES Y LA VARIABLE DEPENDIENTE

```python
# Relación entre edad y cargos del seguro
sns.scatterplot(x='AGE', y='Insurance_Charges', data=data)
plt.title('Relación entre Edad y Cargos del Seguro')
plt.xlabel('Edad')
plt.ylabel('Cargos del Seguro')
plt.show()

# Relación entre BMI y cargos del seguro
sns.scatterplot(x='Body_Mass_Index(BMI)', y='Insurance_Charges', data=data)
plt.title('Relación entre BMI y Cargos del Seguro')
plt.xlabel('BMI')
plt.ylabel('Cargos del Seguro')
plt.show()

# Boxplot de cargos del seguro por género
sns.boxplot(x='Gender', y='Insurance_Charges', data=data)
plt.title('Cargos del Seguro por Género')
plt.xlabel('Género')
plt.ylabel('Cargos del Seguro')
plt.show()

# Boxplot de cargos del seguro por estado de tabaquismo
sns.boxplot(x='Smoking_Status', y='Insurance_Charges', data=data)
plt.title('Cargos del Seguro por Estado de Tabaquismo')
plt.xlabel('Estado de Tabaquismo')
plt.ylabel('Cargos del Seguro')
plt.show()

# Boxplot de cargos del seguro por región
sns.boxplot(x='Region', y='Insurance_Charges', data=data)
plt.title('Cargos del Seguro por Región')
plt.xlabel('Región')
plt.ylabel('Cargos del Seguro')
plt.show()
```









Preparación de datos

- Eliminación de datos duplicados.
- Codificación de variables - One Hot Encoding
- Normalización de variables

Eliminación de datos

4.1 Eliminar datos duplicados.

```
data.drop_duplicates()
```

Hallazgos:
Antes: 1407 filas
Depué:1339 filas.
Se eliminaron 68 filas duplicadas.

Codificación One-Hot-Encoding 1/2

```python
# One Hot Encoder
# Seleccionar columnas categóricas
categorical_columns = data.select_dtypes(include=['object']).columns.tolist()

# Codificar las columnas categóricas
encoder = OneHotEncoder(sparse_output=False)
one_hot_encoded = encoder.fit_transform(data[
one_hot_data = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(categorical_columns))

# Concatenar las columnas codificadas con las otras columnas
data_encoded = pd.concat([data.drop(columns=categorical_columns), one_hot_data], axis=1)

# Mostrar el DataFrame resultante
print(data_encoded)
```

columns: Axes | None

Codificación One-Hot-Encoding 2/2

- **Antes de Codificación:** El DataFrame original tenía 7 columnas.
- **Después de Codificación:** El DataFrame resultante tiene 4 columnas numéricas originales y 8 columnas codificadas, lo que suma un total de 12 columnas.

Gender_female	Gender_male	Smoking_Status_no	Smoking_Status_yes	\
1.0	0.0	0.0	1.0	
0.0	1.0	1.0	0.0	
0.0	1.0	1.0	0.0	
0.0	1.0	1.0	0.0	
0.0	1.0	1.0	0.0	
...	...	...	...	
1.0	0.0	1.0	0.0	
1.0	0.0	1.0	0.0	
1.0	0.0	1.0	0.0	
1.0	0.0	1.0	0.0	
0.0	1.0	1.0	0.0	

Region_northeast	Region_northwest	Region_southeast	Region_southwest
0.0	0.0	0.0	1.0
0.0	0.0	1.0	0.0
0.0	0.0	1.0	0.0
0.0	1.0	0.0	0.0
0.0	1.0	0.0	0.0
...	...	...	...
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0
0.0	1.0	0.0	0.0
0.0	1.0	0.0	0.0
0.0	1.0	0.0	0.0

Escalado de variables numéricas

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
num_vars = ['AGE', 'Body_Mass_Index(BMI)', 'Number_of_Children']
data_encoded[num_vars] = scaler.fit_transform(data_encoded[num_vars])
#Verificar
print(data_encoded.head())
```

```
        AGE  Body_Mass_Index(BMI)  Number_of_Children  Insurance_Charges  \
0 -1.440270             -0.449394           -0.904693        16884.92400
1 -1.511352              0.509676           -0.082404         1725.55230
2 -0.800537              0.383870            1.562174         4449.46200
3 -0.445130             -1.298180           -0.904693        21984.47061
4 -0.516212             -0.289277           -0.904693         3866.85520
```

Métodos predictivos 1/4

- Datos de entranamiento y pruebas.
- Método 1: Regresión Lineal
- Método 2: Random Fores
- Máquinas de soporte vectorial

5.1 División en conjuntos de entranamiento y prueba.

```
[19] from sklearn.model_selection import train_test_split


     X = data_encoded.drop('Insurance_Charges', axis=1)
     y = data_encoded['Insurance_Charges']

     # Dividir los datos en conjuntos de entrenamiento y prueba
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     # Mostrar las formas de los conjuntos para verificar
     print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(1125, 11) (282, 11) (1125,) (282,)
```

Métodos predictivos 2/4

Regresión lineal

5.2 Método 1: Regresión Lineal.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Crear el modelo de regresión lineal
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)

# Hacer predicciones con los datos de prueba
y_pred_lr = model_lr.predict(X_test)

# Evaluar el modelo
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print(f'Regresión Lineal - Mean Squared Error: {mse_lr}')
print(f'Regresión Lineal - R-squared: {r2_lr}')
```

```
Regresión Lineal - Mean Squared Error: 37666354.80802747
Regresión Lineal - R-squared: 0.7895151998242048
```

Métodos predictivos 3/4

Random Forest

```
# Crear el modelo base de bosques aleatorios
model_rf = RandomForestRegressor(random_state=42)
param_grid = {
    'n_estimators': [100, 200],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True, False]
}
# Configurar GridSearchCV
grid_search = GridSearchCV(estimator=model_rf, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
# Medir el tiempo de inicio
start_time = time.time()
# Ajustar el modelo con GridSearchCV
grid_search.fit(X_train, y_train)
# Medir el tiempo de finalización
end_time = time.time()
elapsed_time = end_time - start_time
# Obtener el mejor modelo
best_rf = grid_search.best_estimator_
# Hacer predicciones con los datos de prueba usando el mejor modelo
y_pred_rf = best_rf.predict(X_test)
# Evaluar el modelo
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
```

Métodos predictivos 4/4

Maquina soporte vectorial

```
# Definir el grid de hiperparámetros para GridSearchCV
param_grid = {
    'kernel': ['linear', 'rbf', 'poly'],
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto']
}

# Configurar GridSearchCV
grid_search = GridSearchCV(SVR(), param_grid, cv=3, n_jobs=-1, verbose=2, scoring='neg_mean_squared_erro

# Ajustar el modelo con GridSearchCV
grid_search.fit(X_train, y_train)

# Obtener el mejor modelo
best_svm = grid_search.best_estimator_

# Hacer predicciones con los datos de prueba usando el mejor modelo
y_pred_svm = best_svm.predict(X_test)

# Evaluar el mejor modelo
mse_svm = mean_squared_error(y_test, y_pred_svm)
r2_svm = r2_score(y_test, y_pred_svm)
```

4. Evaluación de Modelos

Para evaluar los modelos utilizamos las siguientes métricas de medición.

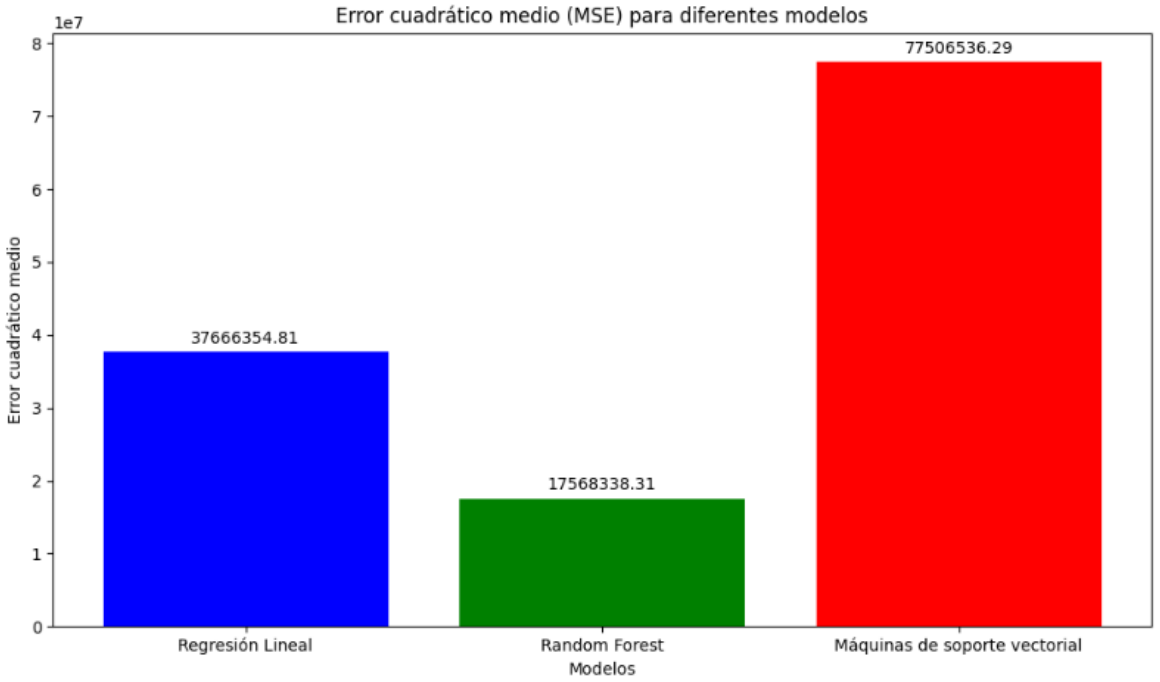
- Error cuadrático medio – MSE
- Coeficiente de determinación - R^2

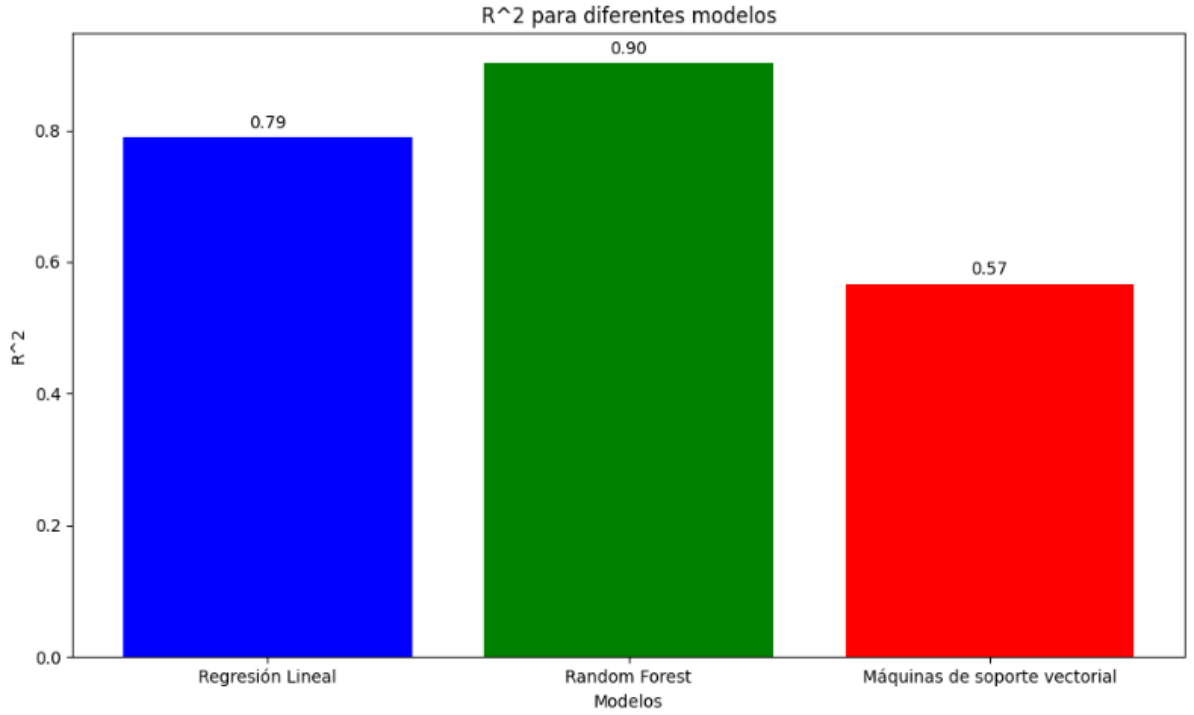
Evaluación de los modelos:

Modelo	Evaluar el modelo	Resultados
Regresión Lineal	mse_lr = mean_squared_error(y_test, y_pred_lr) r2_lr = r2_score(y_test, y_pred_lr)	MSE: 37666354.80802747 R^2: 0.7895151998242048
Random Forest	mse_rf = mean_squared_error(y_test, y_pred_rf) r2_rf = r2_score(y_test, y_pred_rf)	MSE: 17568338.311895933 R^2: 0.901825695694542
Máquinas de soporte vectorial SVM.	mse_svm = mean_squared_error(y_test, y_pred_svm) r2_svm = r2_score(y_test, y_pred_svm)	MSE: 77506536.2894743 R^2: 0.5668827555425889

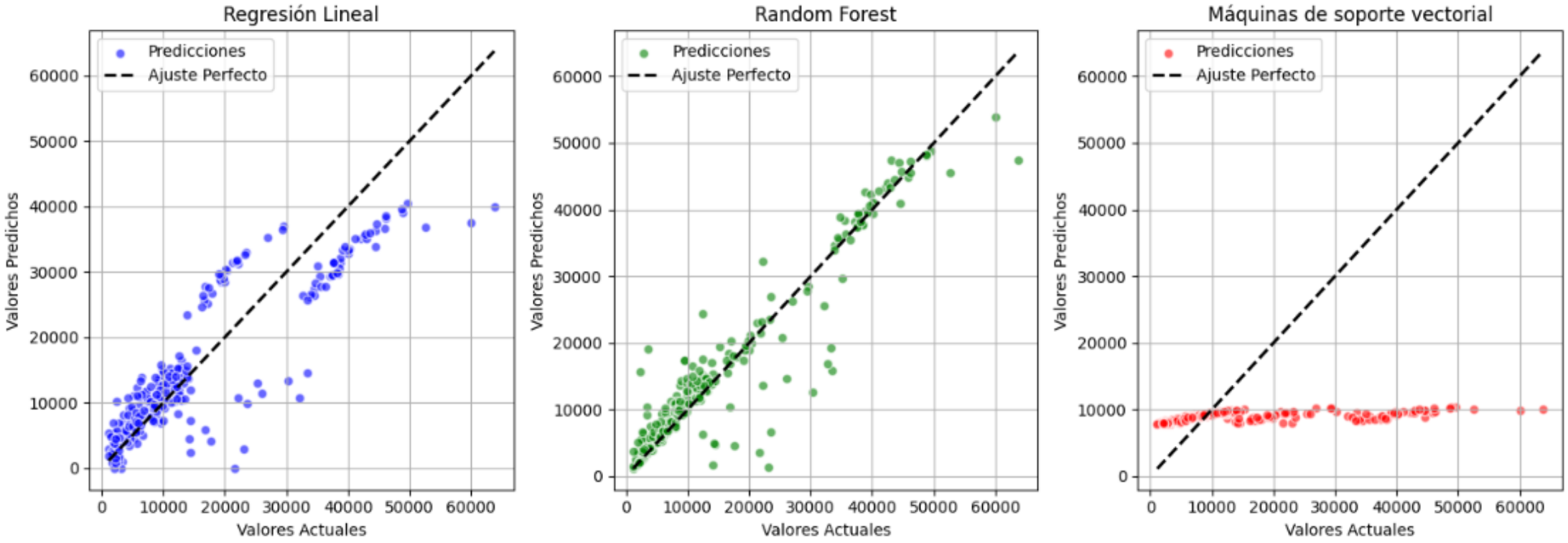
4. Evaluación de Modelos

Comparación de las métricas en los diferentes modelos.





5. Interpretación de Resultados



Modelo de Random Forest:
Este modelo proporciona las predicciones más cercanas a los valores reales, captura mejor la media y la desviación estándar de los datos, además maneja mejor los valores extremos y tiene una mayor precisión.

GRACIAS

UTPL