

PREDICCIÓN DATOS CALIDAD AIRE VALLE DE ABURRÁ

Juan José Naranjo Velasquez
Diego Alejandro Mora Suarez
Kenneth David Leonel Triana



Problemática estudio

La calidad del Aire en el Valle de Aburra es un papel determinante para la población sensible (niños, adultos mayores, personas con que sufran enfermedades respiratorias), el SIATA entidad que hace supervisión y medición de la calidad del aire dentro, de las métricas está el número de 2.5 micras (PM2.5) es de gran relevancia dado que se toma como patrón para medir las partículas con un tamaño minúsculo y penetran más fácil en órganos como los pulmones y perjudicar dicha población.

Objetivo: Predecir el número de partículas PM2.5 en la calidad del aire.


Exploración datos

** Desde: Hasta:

** La información del mes en curso estará disponible para descarga al inicio del próximo mes.

Elige la variable que quieres consultar:

Humedad Precipitación Presión Temperatura Viento Radiación

Buscar :

Seleccione estación(es):

☐ Seleccionar Todas

☐ 1 - Casa de Gobierno Altavista

☐ 2 - Escuela Rural La Verde

☐ 3 - Escuela Rural Yarumalito

☐ 4 - I.E Hector Rogelio Montoya

☐ 5 - I.E Santa Elena

☐ 7 - Escuela Republica de Cuba

☐ 8 - Escuela CEDEPRO



	fecha_hora	temperatura		fecha_hora	codigoserial	pm25	pm10	pm1	no	no2
28_Itagui	0	2024-01-01 00:00:00	19.9	0	2024-01-01 00:00:00	69	36.4093	-9999.0	-9999.0	-9999.0
69_Caldas	1	2024-01-01 00:01:00	19.9	1	2024-01-01 01:00:00	69	68.4104	-9999.0	-9999.0	-9999.0
83_Belen	2	2024-01-01 00:02:00	19.9	2	2024-01-01 02:00:00	69	64.5992	-9999.0	-9999.0	-9999.0
86_Aranjuez	3	2024-01-01 00:03:00	19.9							
	4	2024-01-01 00:04:00	19.9							

```
columnasAEliminar = ['calidad_pm25','calidad_pm10','calidad_pm1','calidad_no','calidad_p_ssr',  
                      'calidad_pliquida_ssr','calidad_rglocal_ssr','calidad_taire10_ssr',  
                      'calidad_vviento_ssr','calidad_no2','calidad_nox','calidad_pst','calidad_dviento_ssr',  
                      'calidad_haire10_ssr','calidad_ozono','calidad_so2','calidad_co']
```

#Se considero que para la variable pm25 se eliminaban ya que no deberia existir valores menores a 0 de esta variable

```
cal_aire_ita = eliminarRegistrosNulos(cal_aire_ita,'pm25', 0)  
cal_aire_cal = eliminarRegistrosNulos(cal_aire_cal,'pm25', 0) # valores menor que 0  
cal_aire_ara = eliminarRegistrosNulos(cal_aire_ara,'pm25', 0)  
cal_aire_bel = eliminarRegistrosNulos(cal_aire_bel,'pm25', 0)
```

```
temp_ita=get_data(temp_ita)  
humedad_ita=get_data(humedad_ita)  
presion_ita=get_data(presion_ita)  
precipitacion_ita=get_data(precipitacion_ita)  
vientos_ita=get_data(vientos_ita)
```


Limpieza del DataSet

```
# por el método knn-imputer
Tabnine | Edit | Test | Explain | Document | Ask
def inputacionDatos(data,columna):

    # Construimos el modelo con 15 vecinos y el peso es uniform
    imputer = KNNImputer(n_neighbors=15, weights="uniform")

    # Ajustamos el modelo e imputamos los missing values
    imputer.fit(data[[columna]])
    data[columna] = imputer.transform(data[[columna]]).ravel()
    # Se rectifica que esos datos NaN ya no existan para rectificar la imputación
    print("Valores perdidos en normalized-losses: " +
          str(data[columna].isnull().sum()))
    return data
```

Se aplico a cada dataframe de cada variable meteorológica una función para pasar valores negativos (-999.0) a NaN ya que son datos que los sensores no registro.

Se hizo la imputación de datos, se usó KnnImputer donde los valores NaN se estableciera un valor con 15 vecinos y pesos uniformes para cada registro

Se guardaron los resultados de la imputación para cada dataframe en un CSV por temas de tiempo en ejecución.

Limpieza de DataSet

1. Se reviso que los dataframes de calidad aire por cada estación solo registraba datos en la partícula contaminante y de estudio con nombre **pm25**, por lo tanto, filtrábamos esos dataframes solo para que quedará lo que es la **fecha_hora**, el **codigoserial** y la partícula ya mencionada.
2. Se hizo el proceso de agrupar en cada dataframe (calidad aire y variables meteorológicas), Esto debido a que se generó nuevas columnas a partir de la fecha_hora como lo es el año, día, mes ,hora. Esa agrupación se ejerció una mediana a partir de la variable de interés en cada dataframe (Temperatura, Humedad,Presion,Públimetro,PM25,Vientos)
3. Se creo una nueva columna llamada **dia_semana**, esta para tener una variable categórica sobre si ese registro fue en el **inicio semana** , **mediana semana** o **fin de semana**.
4. Se creó a partir de la librería Holidays usando el método holidays.CO() para saber si ese día fue o no Festivo y creamos esa columna binaria (0,1)
5. A nuestra variable **codigoSerial** se le hizo un cambio para tener una columna llamada **estacion**, esta nos indica ese registro de que estación pertenece ejemplo **Estacion Caldas**.
6. Se creo una columna a partir de la hora de cada registro para saber a qué franja horaria pertenece (**Madrugada,Mañana,Tarde,Noche**).
7. Se concateno todos los 24 dataframes de nuestras cuatro estaciones.

```
def ConcatenadoRegistros(*data):  
    """  
    Función para concatenar dataframes  
    In[0]: (dataset1, dataset2, dataset3, ..., datasetn)  
    Out[0]: dataset concatenado  
    """  
  
    return pd.concat([*data], axis=0)  
  
# Si queremos unir el de itagui calidadAireItaguiUnido  
dataSetCalidadAire = ConcatenadoRegistros[calidadAireItaguiUnido,calidadAireCaldasUnido,  
[calidadAireAranjuezUnido, calidadAireBelenUnido]  
  
dataSetCalidadAire.head(3)
```



Complejidad de la Limpieza

1. Se realizó la eliminación de atípicos del dataset concatenado mediante los Intercuantiles y mediante el uso de LocalOutlierFactor (con 5 vecinos y el contaminante de 5%).
2. Se realizó a las columnas categóricas one hot encoding.
3. Se estandarizaron los datos y se guardó el scaler

Complejidades observadas

1. Imputación (No nos dio efectividad tener el DataSet con la imputación dataSetLimpio.csv)
2. Inconvenientes para determinar la granularidad

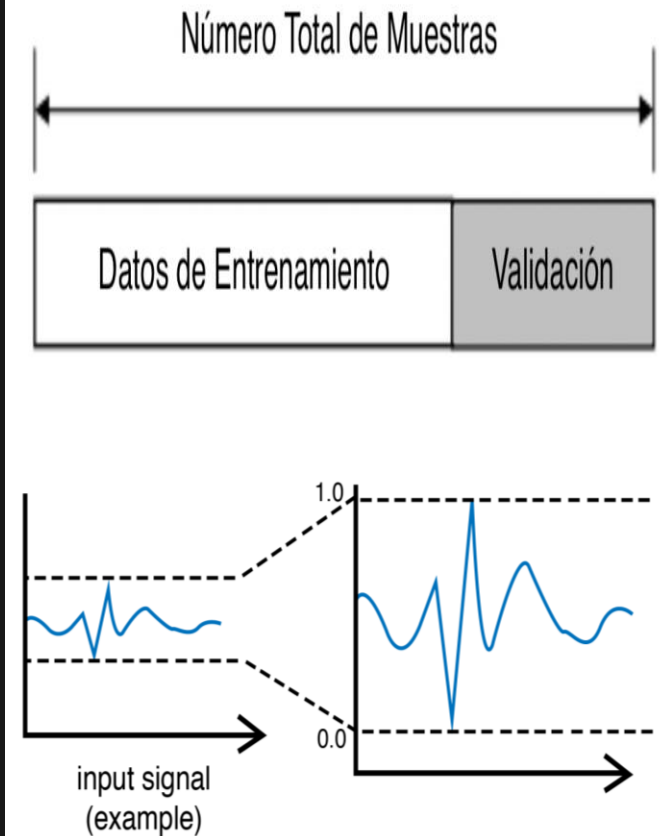
Finalización

1. Con los procesos ya mencionados obtuvimos el DataSet de Limpieza más óptimo con el nombre dataSetLimpia4.csv

Generalidades en modelos de regresión

```
[ 'anio',  
  'mes',  
  'dia',  
  'hora',  
  'festivo',  
  'temperatura',  
  'humedad',  
  'presion',  
  'p1',  
  'velocidad_prom',  
  'velocidad_max',  
  'direccion_prom',  
  'direccion_max',  
  'dia_semana_Fin de Semana',  
  'dia_semana_Inicio Semana',  
  'dia_semana_Media semana',  
  'estacion_Estacion Aranjuez',  
  'estacion_Estacion Belen',  
  'estacion_Estacion Caldas',  
  'estacion_Estacion Itagui',  
  'franja_horaria_Madrugada',  
  'franja_horaria_Mañana',  
  'franja_horaria_Noche',  
  'franja_horaria_Tarde']  
  
y = limpiaDataSetCalidadAire['pm25']
```

```
# División de los datos en datos de entrenamiento del modelo, y datos para el testeo  
X_train, X_test, y_train, y_test = train_test_split(  
    X,  
    y,  
    train_size=0.7, # 70% de nuestros datos en entrenamiento y 30% de los datos para el testeo  
    random_state=1234,  
    shuffle=True  
)  
  
#Escalar Variables numéricas  
pd.set_option('display.float_format', lambda x: '%.4f' % x)  
scaler = joblib.load('data/stage/estandarizacionDataSet4.pkl')  
  
# Se escalan los valores del dataset entrenamiento y prueba  
X_train[x_cols] = scaler.fit_transform(X_train[x_cols])  
X_test[x_cols] = scaler.transform(X_test[x_cols])
```



Maquina de Soporte Vectorial (SVM)

```
# Probamos diferentes kernel con hiperparametros para
# revisar sus evaluaciones
SVMR_linear = SVR(kernel='linear', C = 1000)
SVMR_Pol2 = SVR(kernel='poly', C = 100, degree = 2)
SVMR_Pol3 = SVR(kernel='poly', C = 100, degree = 3)
SVMR_rbf = SVR(kernel='rbf', C = 100)
```

```
Linear   Training: 0.1773573465251853 Test: 0.18930319587891553
Poly 2   Training: 0.3047927153853718 Test: 0.2998748858889463
Poly 3   Training: 0.4229661581884916 Test: 0.38931370249483677
rbf      Training: 0.4465240185131295 Test: 0.376663326638621
```

```
GridSearchCV  ⓘ ⓘ
best_estimator_: SVR
SVR  ⓘ
SVR(C=1, gamma=1, kernel='poly')
```

```
# Se crea un modelo SVR con los mejores hiperparámetros
modelsvr = SVR( kernel = best_params["kernel"]
                 , gamma = best_params["gamma"]
                 , C= best_params["C"]
                 , degree = best_params["degree"])
```

```
#Se entrena el modelo con los datos completos
modelsvr.fit(X_completo, y_completo)
```

```
SVR  ⓘ ⓘ
SVR(C=1, gamma=1, kernel='poly')
```

	param_C	param_degree	param_kernel	param_gamma	mean_test_score	std_test_score	scoreWithStd
14	1.0000	3.0000	poly	1	0.3720	0.0130	28.5417
22	1.0000	NaN	rbf	1	0.2727	0.0116	23.5500
6	0.1000	3.0000	poly	1	0.3173	0.0152	20.9040

Árboles de Decisión

- Modelo inicial

```
#Creacion del Modelo de Arbole de Regresion
modelo=DecisionTreeRegressor(
    max_depth=3,
    random_state=123
)
#Entrenamiento del Modelo
modelo.fit(X_train,y_train)
```

```
#imprimos los valores del R2
print('Training',modelo.score(X_train,y_train))
print('Test      :',r2_score(y_test,modelo.predict(X_test)))
```

```
Training 0.29324600909551024
Test      : 0.284468987942032
```

- Identificación de Mejores Hiperparametros

```
# Se procede a entrenar el modelo con varios multiparametros para buscar el mejor

modeloTree=DecisionTreeRegressor()
CV=10

# Se establecen los hiperparametros
parametros={'max_depth': [2, 4, 6, 8,10,15,20],
            'min_samples_split': [2, 5, 10, 15],
            'min_samples_leaf': [2, 5, 10, 15, 20],
            'max_features': [None, 'sqrt', 'log2']}

#Definimos un GridSearchCV con validacion cruzada

validacion=GridSearchCV(
    modeloTree,
    param_grid=parametros,
    cv=CV,
    verbose=3)

# Se entrena el GridSearchSV
validacion.fit(X_completo,y_completo)
```

Árboles de Decisión

- Mejores Hiperparametros:

```
# Se identifican los mejores hiperparametros
mejoresParametros=validacion.best_params_
print('Mejores Hiperparametros: ',mejoresParametros)
```

```
Mejores Hiperparametros: {'max_depth': 6, 'max_features': None, 'min_samples_leaf': 5, 'min_samples_split': 5}
```

- Modelo con mejores Hiperparametros:

```
print(f'El resultado del modelo con los mejores hiperparametros es de:{modeloTreeR.score(x_completo,y_completo)}')
```

```
El resultado del modelo con los mejores hiperparametros es de:0.4349502870772104
```

Random Forest

Modelo Inicial

```
Click to add a breakpoint 10
modelRF = RandomForestRegressor(
    n_estimators = 25, #Con tanteo s
    criterion = 'squared_error',
    max_depth = None,
    max_features = 'sqrt',
    oob_score = False, #out-of-
    n_jobs = -1,
    random_state = 123
)
# Entrenamiento del modelo
modelRF.fit(X_train, y_train)
```

El error (rmse) de test es:

6.9178

El error (r2) de test es:

0.5385

Proceso para hallar los HiperParametros

```
modelRF = RandomForestRegressor(random_state = 123)
CV = 10
parameters = {
    'n_estimators': [15,20,25],
    'max_features': [6,8,10,12,14],
    'max_depth' : [4,8,15]
}

grid_RF = GridSearchCV(
    estimator = modelRF,
    param_grid = parameters,
    scoring = 'r2',
    n_jobs = -1,
    cv = CV,
    refit = True,
    verbose = 3,
    return_train_score = True
)

grid_RF.fit(X = X_Completo, y = y_Completo)
```

Mejores Parámetros

{'max_depth': 15, 'max_features': 14,
'n_estimators': 25}

Random Forest

Modelo Final

```
# Se crea un modelo Random con los mejores hiperparámetros
modeloFinalRF = RandomForestRegressor( max_depth = int(results_grid_RF_filtered.loc
[indice_max_scoreWithStd]["param_max_depth"])
, max_features = int(results_grid_RF_filtered.loc
[indice_max_scoreWithStd]["param_max_features"])
, n_estimators = int(results_grid_RF_filtered.loc
[indice_max_scoreWithStd]["param_n_estimators"])
, random_state = 1234)
#Se entrena el modelo con los datos completos
modeloFinalRF.fit(X_train, y_train)
```

El error (rmse) de test es: 7.5100

El error (r2) de test es: 0.4562

Regresión Lineal

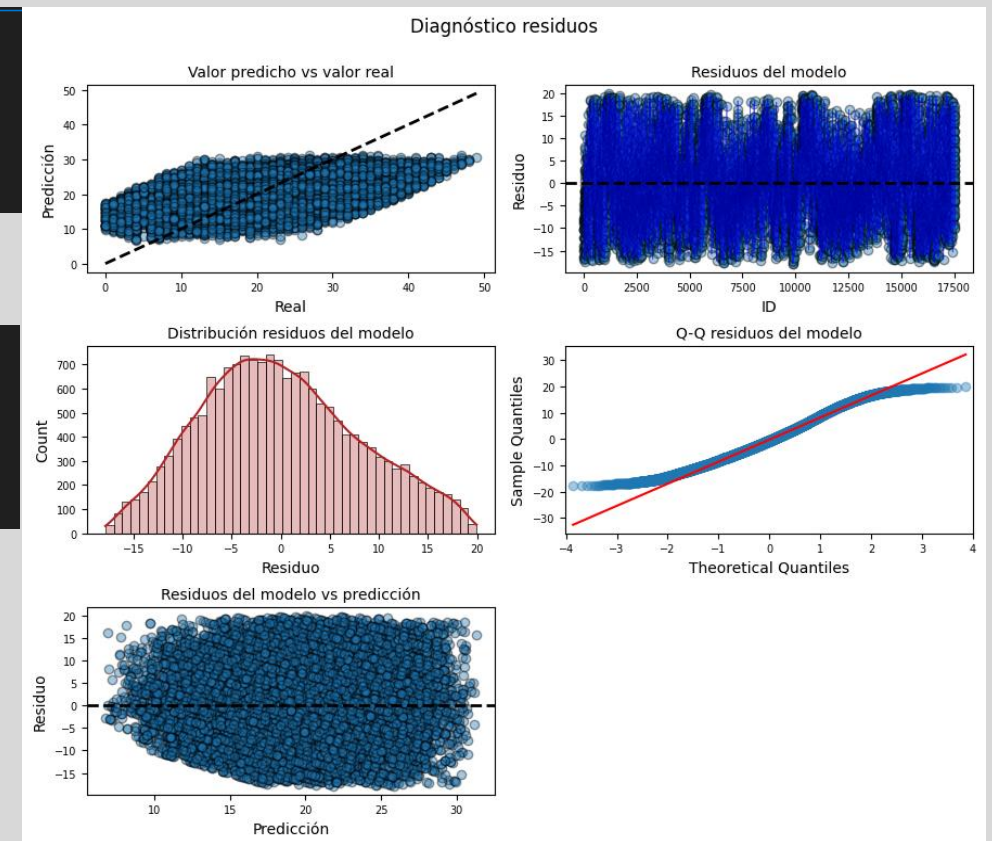
Modelo Inicial

OLS Regression Results			
=====			
Dep. Variable:	pm25	R-squared:	0.193
Model:	OLS	Adj. R-squared:	0.192
Method:	Least Squares	F-statistic:	244.0
Date:	Thu, 24 Oct 2024	Prob (F-statistic):	0.00

Modelo Final

OLS Regression Results			
=====			
Dep. Variable:	pm25	R-squared:	0.235
Model:	OLS	Adj. R-squared:	0.234
Method:	Least Squares	F-statistic:	282.9

1. Se hizo eliminación de valores atípicos por diferentes Técnicas
2. Multicolinealidad de las variables
3. Validación de supuestos



Evaluación Modelos.

Se evalúan los modelos Árboles de Decisión, Random Forest, Máquina de Vectores de Soporte y Regresión Lineal

	model	Descripcion	r2_score	mae	rmse
0	ModeloArbol_PM25_CV.pkl	DecisionTreeRegressor(max_depth=6, min_samples...	0.4350	5.9642	7.6289
1	modeloRandomForestPM25.pkl	(DecisionTreeRegressor(max_depth=8, max_featur...	0.4861	5.7377	7.2756
2	SVR_CV_varios_cal_aire_pm25.pkl	SVR(C=1, gamma=1, kernel='poly')	0.4108	5.9171	7.7902


```
# Top n de los mejores modelos  
dr.sort_values(by='r2_score', ascending=False)
```

✓ 0.0s

	model	Descripcion	r2_score	mae	rmse
1	modeloRandomForestPM25.pkl	(DecisionTreeRegressor(max_depth=8, max_featur...	0.4861	5.7377	7.2756
0	ModeloArbol_PM25_CV.pkl	DecisionTreeRegressor(max_depth=6, min_samples...	0.4350	5.9642	7.6289
2	SVR_CV_varios_cal_aire_pm25.pkl	SVR(C=1, gamma=1, kernel='poly')	0.4108	5.9171	7.7902

Conclusiones:

- El modelo del Arbol de Decisión logró un R^2 de 0.435, lo que explica aproximadamente el 43.5% de la varianza en los niveles de PM2.5, lo cual es relativamente bajo.
- El modelo RandomForest tiene un R^2 de 0.4861, mejorando un poco respecto al Árbol de Decisión, lo que sugiere que los árboles adicionales han capturado mejor la variabilidad de los datos.
- El modelo de máquinas de soporte vectorial con un R^2 de 0.418 no alcanza una precisión acertada y es similar al Árbol de Decisión, aunque las máquinas de soporte vectorial pueden ser efectivas en problemas no lineales, en este caso no supera al bosque aleatorio.

Conclusiones:

- Dado que los modelos con la variable de salida de PM25 tienen un R^2 inferior al 0.6, exploramos los datos hacia la variable temperatura, obteniendo estos resultados:

	model	Descripcion	r2_score	mae	rmse
0	ModeloArbolTemperatura_CV.pkl	DecisionTreeRegressor(max_depth=10, min_sample...	0.9494	0.6675	0.8527
1	modeloRandomForestTemperatura.pkl	(DecisionTreeRegressor(max_depth=15, max_featu...	0.9782	0.4134	0.5596
2	SVR_CV_varios_cal_aire_temp.pkl	SVR(C=1, gamma=1, kernel='poly')	0.9552	0.6142	0.8018

- Se observa que los datos se comportan de una mejor manera en base a la Temperatura aumentando sus R^2 hasta en un 0.9782 para el Caso del Random Forest, siguiendo un comportamiento similar para los modelos de Arboles de decisión y Maquina de soporte vectorial. Esto relacionado con la matriz de correlación entre los factores.