

VVB en Australia

Examen Transversal

Integrantes:

Aldo Guasch

Marco Calvillán

Raúl Concha

Tabla de contenido

Descripción del contexto	4
Propósito y justificación del proyecto.....	5
Análisis del Data set Entregado	6
Carga del Data set	8
Información preliminar de los datos.....	8
Dimensiones del DataSet y Tipos de Datos.....	9
Análisis por registros aleatorios	10
Análisis por estructura del Data Set	11
Datos nulos dentro del Dataset.....	12
Estadísticas generales.....	13
Hallazgo 1:.....	15
Hallazgo 2:.....	16
Hallazgo 3:.....	17
Limpieza de valores nulos.....	18
Estandarización de datos	20
Realizar razonamiento y justificación respecto del análisis de los datos	24
Selección de modelos a utilizar.....	24
Implementación de modelos	25
Modelos de Clasificación.....	25
Random Forest Classifier.....	26
SVC.....	28
K-Neighbors Classifier	29
Modelos de Regresión.....	30
Linear Regresión	32
SVR.....	32
GradientBoostingRegressor.....	33
Random Forest Regressor.....	33
K-NeighborsRegressor	34
Evaluación de modelos	35
Fases	37
Toma de decisiones.....	38
Generación de CSV	38

Gráficos con POWER BI	38
Gráficos Interactivos	39
Ranking de temperaturas por ciudad	41
Ranking de Humedad por ciudad	42
Mapa del promedio de caída de lluvia en Australia	43
Gráfico del porcentaje cantidad de lluvias por ciudad.....	44
Gráficos Estáticos.....	45
Grafica del porcentaje de Lluvias.....	46
Tendencia Caída de lluvia.....	47
Ranking Dirección del viento	47
Gráfico de Tendencia Temperatura Máxima.....	48
Conclusión	49

Descripción del contexto

Dado a que Australia se trata de un país seco, la gran cantidad de las ciudades del territorio sufren de sequía o en el mejor de los casos de precipitaciones moderadas, contando en general con inviernos ligeros y veranos cálidos, dado a estas condiciones del país se buscan maneras de aprovechar las precipitaciones por escasas que sean, por lo que se nos ha solicitado realizar un análisis de datos para realizar predicciones con la minería de datos.

Para esto se nos ha entregado un set de datos perteneciente a observaciones meteorológicas diarias de múltiples ubicaciones en Australia, obtenidas de la Oficina de Meteorología de Commonwealth en Australia y estas fueron procesadas para crear este conjunto de datos de muestra, entre los datos procesados se ha proporcionado una variable objetivo llamada RainTomorrow y esta se divide entre resultados No/Sí para indicar que si habrá lluvia el día del registro adicionalmente, existe una variable de riesgo nombrada RISK_MM que indica la lluvia registrada en milímetros. Todos estos datos se encuentran en un archivo CSV que utilizaremos para extraer y desplegar dicha información en un proyecto de minería con Jupyter.

Como objetivo debemos utilizar estos datos que se nos han entregado para poder crear un modelo capaz de predecir en qué lugares de Australia habrá lluvias en los próximos días, para dar paso a esto llevaremos a cabo esta primera etapa en la que ordenaremos los datos que nos han entregado mediante la carga, integración y limpieza de estos haciéndolos más legibles para el cliente, llevando esto a cabo en las siguientes etapas que son la Comprensión de los datos y su Preparación.

Empresas que buscan establecerse en Australia con distintos fines, podrían necesitar la información con datos relacionados al clima.

Propósito y justificación del proyecto

La vitivinícola VVB es una empresa familiar chilena que se especializa en hacer vinos de distintas cepas, las cuales comprenden: Chardonnay, Sauvignon Blanc, Cabernet Sauvignon, entre otros. Están buscando expandirse al extranjero y llegaron a la decisión de instalarse en Australia.

Por lo que, nos encomendaron la tarea de encontrar una de las mejores zonas para poder instalar un viñedo. Por lo que investigamos las condiciones perfectas en la que se deben plantar las uvas.

Clima: Se deben descartar todas las regiones demasiado húmedas, de clima lluvioso y poca insolación. Las mejores son las regiones de clima seco y árido, con escasa lluvias anuales que se concentren sobre todo a fines del otoño y en el invierno. Otro factor importante, para la calidad de los viñedos, es la amplitud térmica, abundante sol durante el día y fresco durante la noche. Esta variación de temperatura facilita la formación de las sustancias aromáticas y ayuda a la fijación de pigmentos responsables del color y el desarrollo de los componentes que otorgan cuerpo y estructura al vino. Los vinos resultantes son de grandes cualidades aromáticas, de buen color y mucho cuerpo, condiciones imprescindibles para que un vino sea “Añejable”.

Altitud: La mayoría de los viñedos se encuentran entre los 350 y 2300 m. sobre el nivel del mar. A mayor altitud aumenta la amplitud térmica y mejora la radiación solar, favoreciendo esa maduración lenta de la que ya hablamos. Las zonas más altas se encuentran alejadas de las grandes poblaciones y por ende de la contaminación produciendo una materia prima especialmente sana.

Análisis del Data set Entregado

Como primer paso dentro de la Comprensión de los datos daremos un vistazo al conjunto de datos que nos han sido entregados para hacer un análisis general y observar los posibles datos relevantes y problemas presentes que puedan obstaculizar la implementación de nuestra solución:

En primer lugar, cabe mencionar que estas son las columnas que posee nuestro data set para tener un mejor entendimiento

Columna	Descripción
Fecha	fecha de la observación
Ubicación	ubicación de la estación meteorológica
MinTemp	temperatura mínima en grados Celsius
MaxTemp	temperatura máxima en grados Celsius
Lluvia	cantidad de lluvia registrada ese día en mm.
Evaporacion	evaporación (mm) en 24 horas
Sol	número de horas de sol brillante en el día
DirRafaga	dirección de la ráfaga de viento más fuerte en 24 horas.
VelRafaga	velocidad (km/hr) de la ráfaga de viento más fuerte en 24 horas.
Dir9am	dirección del viento a las 9am
Dir3pm	dirección del viento a las 3pm
Vel9am	velocidad (km/hr) del viento a las 9am

Vel3pm	velocidad (km/hr) del viento a las 3pm
Hum9am	porcentaje de humedad a las 9am
Hum3pm	porcentaje de humedad a las 3pm
Pres9am	presión atmosférica (hpa) a nivel del mar a las 9am
Pre3pm	presión atmosférica (hpa) a nivel del mar a las 3pm
Nub9am	fracción del cielo cubierto por nubes a las 9am. Se mide en "octavos", de manera que un valor 0 indica cielo totalmente despejado y 8, cielo totalmente cubierto.
Nub3pm	fracción del cielo cubierto por nubes a las 3pm. Se mide en "octavos", de manera que un valor 0 indica cielo totalmente despejado y 8, cielo totalmente cubierto.
Temp9am	temperatura en grados celsius a las 9am
Temp3pm	temperatura en grados celsius a las 3pm
LluviaHoy	variable indicadora que toma el valor 1 si la precipitación en mm. en las últimas 24 hrs. excede 1 mm. y 0 si no.
RISK_MM	La cantidad de lluvia. Una especie de medida del "riesgo".
LluviaMan	variable indicadora que toma el valor 1 si al día siguiente llovió y 0 si no.

Luego de observar a que hace referencia cada columna podremos entender mejor los datos que se presentan en cada una de estas:

	A	B	C	D	E	F	G	H	I	J	K	L	
1	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	
2	2008-12-01	Albury	13.4	22.9	0.6	NA	NA	W	44	W	WNW	20	
3	2008-12-02	Albury	7.4	25.1	0	NA	NA	WNW	44	NNW	WSW	4	
4	2008-12-03	Albury	12.9	25.7	0	NA	NA	WSW	46	W	WSW	19	
5	2008-12-04	Albury	9.2	28	0	NA	NA	NE	24	SE	E	11	
6	2008-12-05	Albury	17.5	32.3		1	NA	NA	W	41	ENE	NW	7
7	2008-12-06	Albury	14.6	29.7	0.2		NA	NA	WNW	56	W	W	19
8	2008-12-07	Albury	14.3	25		0	NA	NA	W	50	SW	W	20
9	2008-12-08	Albury	7.7	26.7		0	NA	NA	W	35	SSE	W	6
10	2008-12-09	Albury	9.7	31.9		0	NA	NA	NNW	80	SE	NW	7
11	2008-12-10	Albury	13.1	30.1		1.4	NA	NA	W	28	S	SSE	15
12	2008-12-11	Albury	13.4	30.4		0	NA	NA	N	30	SSE	ESE	17
13	2008-12-12	Albury	15.9	21.7		2.2	NA	NA	NNE	31	NE	ENE	15
14	2008-12-13	Albury	15.9	18.6		15.6	NA	NA	W	61	NNW	NNW	28
15	2008-12-14	Albury	12.6	21		3.6	NA	NA	SW	44	W	SSW	24
16	2008-12-16	Albury	9.8	27.7	NA		NA	NA	WNW	50	NA	WNW	NA
17	2008-12-17	Albury	14.1	20.9		0	NA	NA	ENE	22	SSW	E	11
18	2008-12-18	Albury	13.5	22.9		16.8	NA	NA	W	63	N	WNW	6
19	2008-12-19	Albury	11.2	22.5		10.6	NA	NA	SSE	43	WSW	SW	24
20	2008-12-20	Albury	9.8	25.6		0	NA	NA	SSE	26	SE	NNW	17
21	2008-12-21	Albury	11.5	29.3		0	NA	NA	S	24	SF	SF	9

M	N	O	P	Q	R	S	T	U	V	W	X	
WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RISK_MM	RainTomorrow	
24	71	22	1007.7	1007.1	8	NA	16.9	21.8	No	0	No	
22	44	25	1010.6	1007.8	NA		17.2	24.3	No	0	No	
26	38	30	1007.6	1008.7	NA		2	21	23.2	No	No	
9	45	16	1017.6	1012.8	NA		18.1	26.5	No	1	No	
20	82	33	1010.8	1006	7	8	17.8	29.7	No	0.2	No	
24	55	23	1009.2	1005.4	NA		20.6	28.9	No	0	No	
24	49	19	1009.6	1008.2	1	NA	18.1	24.6	No	0	No	
17	48	19	1013.4	1010.1	NA		16.3	25.5	No	0	No	
28	42	9	1008.9	1003.6	NA		18.3	30.2	No	1.4	Yes	
11	58	27	1007	1005.7	NA		20.1	28.2	Yes	0	No	
6	48	22	1011.8	1008.7	NA		20.4	28.8	No	2.2	Yes	
13	89	91	1010.5	1004.2	8	8	15.9	17	Yes	15.6	Yes	
28	76	93	994.3	993	8	8	17.4	15.8	Yes	3.6	Yes	
20	65	43	1001.2	1001.8	NA		7	15.8	19.8	Yes	0	No
22	50	28	1013.4	1010.3	0	NA		17.3	26.2	NA	0	No
9	69	82	1012.2	1010.4	8	1	17.2	18.1	No	16.8	Yes	
20	80	65	1005.8	1002.2	8	1	18	21.5	Yes	10.6	Yes	
17	47	32	1009.4	1009.7	NA		2	15.5	21	Yes	0	No
6	45	26	1019.2	1017.1	NA		15.8	23.2	No	0	No	
9	56	28	1019.3	1014.8	NA		19.1	27.3	No	0	No	

Acá tenemos las columnas desde “Date” hasta “RainTomorrow” y cómo podemos apreciar a simple vista ya podemos ver algunas filas con datos desordenados o con

valores Nulos (NA) por lo que ya podemos ir haciéndonos una idea de los datos que necesitaremos organizar o limpiar para la tercera etapa. Para poder apreciar a más detalle el data set y sus columnas iremos al siguiente paso y cargaremos el data set en un notebook Jupyter.

Carga del Data set

Comenzaremos la etapa 2 cargando la base de datos (En este caso un archivo CSV), para comenzar importamos algunas de las librerías que utilizaremos. De las más importantes a trabajar es ‘Pandas’ y Numpy la cual importamos más adelante.

```
[ ] #Importamos la librería de Pandas  
import pandas as pd
```

El Archivo CSV se carga de un repositorio de Git para que todos los que tengan el código lo puedan ejecutar y que no tengan problemas con la carga. Luego se indica el separador que tiene el data set y el cómo se separan los decimales.

```
● ##Colocamos la ruta del drive en una variable para luego pasarlo a un dataframe.  
url = 'https://raw.githubusercontent.com/AldoGuaschHuerta/AustraliaCSV/main/weatherAUS.csv'  
df = pd.read_csv(url, sep=';', decimal= '.')
```

Información preliminar de los datos

Luego de haber importado desde drive el contenido del dataset podemos comenzar a realizar comandos para poder visualizar detalles de este como la información preliminar

```
[ ] #Información preliminar de los datos  
df.info()
```

```
[ ] <class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Date              142193 non-null   object  
 1   Location          142193 non-null   object  
 2   MinTemp           141556 non-null   float64 
 3   MaxTemp           141871 non-null   float64 
 4   Rainfall          140787 non-null   float64 
 5   Evaporation      81350 non-null    float64 
 6   Sunshine          74377 non-null    float64 
 7   WindGustDir       132863 non-null   object  
 8   WindGustSpeed    132923 non-null   float64 
 9   WindDir9am        132180 non-null   object  
 10  WindDir3pm        138415 non-null   object  
 11  WindSpeed9am     140845 non-null   float64 
 12  WindSpeed3pm     139563 non-null   float64 
 13  Humidity9am      140419 non-null   float64 
 14  Humidity3pm      138583 non-null   float64 
 15  Pressure9am      128179 non-null   object  
 16  Pressure3pm      128212 non-null   object  
 17  Cloud9am          88536 non-null   float64 
 18  Cloud3pm          85099 non-null   float64 
 19  Temp9am           141289 non-null   float64 
 20  Temp3pm           139467 non-null   float64 
 21  RainToday         140787 non-null   object  
 22  RISK_MM           142193 non-null   float64 
 23  RainTomorrow     142193 non-null   object  
dtypes: float64(15), object(9)
memory usage: 26.0+ MB
```

Vemos que hay 15 columnas (Características) de tipo float64 y 9 tipo object; 142193 filas (Registros).

Dimensiones del DataSet y Tipos de Datos

Entre otras especificaciones que podemos visualizar están las dimensiones y los tipos de datos del Dataset.

La siguiente línea de código nos enseña las dimensiones del DataSet, la primera información son las filas (Registros) y la segunda son las columnas (Características)

```
[ ] #Dimensiones del DataSet
df.shape
(142193, 24)
```

Podemos ver los tipos de datos que tiene el DataSet con la siguiente línea de código, vemos que tenemos Tipos de datos Object y Float64

```
[ ] #Tipos de datos del DataSet  
df.dtypes
```

```
Date          object  
Location      object  
MinTemp       float64  
MaxTemp       float64  
Rainfall       float64  
Evaporation   float64  
Sunshine       float64  
WindGustDir    object  
WindGustSpeed  float64  
WindDir9am     object  
WindDir3pm     object  
WindSpeed9am   float64  
WindSpeed3pm   float64  
Humidity9am    float64  
Humidity3pm    float64  
Pressure9am    object  
Pressure3pm    object  
Cloud9am       float64  
Cloud3pm       float64  
Temp9am        float64  
Temp3pm        float64  
RainToday      object  
RISK_MM        float64  
RainTomorrow   object  
dtype: object
```

Análisis por registros aleatorios

Una de las funciones que podemos hacer es ver una cierta cantidad asignada de registros aleatorios dentro del dataset, en este caso asignamos 5 a la función `sample()`.

```
[ ] #También se pueden ver registros aleatorios con Sample(x) donde x es la cantidad de ejemplos que queremos ver.  
df.sample(5)
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am
8554	2016-06-06	Cobar	8.6	16.4	0.0	2.0	NaN	W	35.0	WSW	...	55.0	1009.3	1007.4	8.0
40125	2013-05-08	Williamtown	9.5	20.7	0.0	1.6	2.6	SE	26.0	NW	...	67.0	1031.1	1028	3.0
136710	2010-08-29	Darwin	23.4	34.7	0.0	7.0	11.3	ESE	41.0	E	...	30.0	1015.8	1011.6	1.0
49238	2013-04-02	Tuggeranong	3.2	23.8	0.0	NaN	NaN	WNW	24.0	NaN	...	37.0	1016.8	1013.9	NaN
76770	2011-11-16	Watsonia	12.8	20.2	11.2	6.6	11.5	WSW	39.0	SE	...	56.0	1019.4	1019.3	7.0

5 rows × 24 columns

<

Análisis por estructura del Data Set.

Podemos visualizar por estructura al data set, dependiendo de los parámetros que usemos podemos llamarlo completo o los primeros 5 resultados, así como los 5 últimos.

```
[ ] #análisis por estructura del dataSet
#los 5 primeros registros se visualizan con head()
df.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	22.0	1007.7	1007.1	8.0	NaN
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	25.0	1010.6	1007.8	NaN	NaN
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	30.0	1007.6	1008.7	NaN	2.0
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	16.0	1017.6	1012.8	NaN	NaN
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	33.0	1010.8	1006	7.0	8.0

5 rows × 24 columns

Temp9am	Temp3pm	RainToday	RISK_MM	RainTomorrow
16.9	21.8	No	0.0	No
17.2	24.3	No	0.0	No
21.0	23.2	No	0.0	No
18.1	26.5	No	1.0	No
17.8	29.7	No	0.2	No

```
[ ] #Los Últimos 5 registros se visualizan con tail()
df.tail()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm
142188	2017-06-20	Uluru	3.5	21.8	0.0	NaN	NaN	E	31.0	ESE	...	27.0	1024.7	1021.2	NaN	NaN
142189	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0	SE	...	24.0	1024.6	1020.3	NaN	NaN
142190	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0	SE	...	21.0	1023.5	1019.1	NaN	NaN
142191	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0	SE	...	24.0	1021	1016.8	NaN	NaN
142192	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0	SSE	...	24.0	1019.4	1016.5	3.0	NaN

5 rows × 24 columns

Datos nulos dentro del Dataset

También podemos utilizar una función que nos permite filtrar y enumerar los valores nulos encontrados en cada columna del datasets

```
[ ] # La siguiente porción de código nos permite filtrar el data set
# Nos entrega la cantidad de nulos que tiene cada Característica y la ordena del mayor a menor.
list_nulls = df.isnull().sum()
list_nulls[(list_nulls>0)].sort_values(ascending = False)

Sunshine      67816
Evaporation   60843
Cloud3pm       57094
Cloud9am       53657
Pressure9am    14014
Pressure3pm    13981
WindDir9am     10013
WindGustDir    9330
WindGustSpeed   9270
WindDir3pm      3778
Humidity3pm    3610
Temp3pm         2726
WindSpeed3pm    2630
Humidity9am     1774
RainToday        1406
Rainfall         1406
WindSpeed9am    1348
Temp9am          904
MinTemp          637
MaxTemp          322
dtype: int64
```

Apreciamos la cantidad de valores nulos que tiene cada columna, siendo los que tienen mayor cantidad ‘Sunshine’ y ‘Evaporation’.

Estadísticas generales

Para poder conocer la correlación que existe entre las variables, podemos utilizar dos funciones dependiendo del tipo de variable, si es Cuantitativa utilizaremos la matriz de correlación de Pearson, la cual se llama con la función. corr() en python. A medida que el valor se acerca a 1, la correlación es directa y positiva. si es cercana a 0, no tiene correlación. y por contraparte, si es cercana a -1 la correlación es negativa. Esto solo para cuando la asociación de la variable es lineal.

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Cloud9am	Cloud3pm	Temp9am
MinTemp	1.000000	0.736267	0.104255	0.467261	0.072961	0.177285	0.176005	0.175749	-0.234211	0.005999	0.077625	0.020489	0.901813
MaxTemp	0.736267	1.000000	-0.074839	0.588915	0.469967	0.067690	0.014680	0.050800	-0.505432	-0.509270	-0.289865	-0.279053	0.887020
Rainfall	0.104255	-0.074839	1.000000	-0.064549	-0.227525	0.133497	0.086816	0.057759	0.223725	0.255312	0.198195	0.171993	0.011477
Evaporation	0.467261	0.588915	-0.064549	1.000000	0.366607	0.203001	0.193936	0.128895	-0.505890	-0.392785	-0.185032	-0.184287	0.545497
Sunshine	0.072961	0.469967	-0.227525	0.366607	1.000000	-0.032831	0.080840	0.058012	-0.491603	-0.629122	-0.675610	-0.704202	0.291139
WindGustSpeed	0.177285	0.067690	0.133497	0.203001	-0.032831	1.000000	0.604837	0.686419	-0.215461	-0.026663	0.071235	0.109088	0.150258
WindSpeed9am	0.176005	0.014680	0.086816	0.193936	0.008040	0.604837	1.000000	0.519971	-0.270807	-0.031607	0.024280	0.053584	0.129298
WindSpeed3pm	0.175749	0.050800	0.057759	0.128895	0.056012	0.686419	0.519971	1.000000	-0.145942	0.015903	0.052780	0.025269	0.163601
Humidity9am	-0.234211	-0.505432	0.223725	-0.505890	-0.491603	-0.215461	-0.270807	-0.145942	1.000000	0.667388	0.452182	0.358043	-0.472826
Humidity3pm	0.005999	-0.509270	0.255312	-0.392785	-0.629122	-0.026663	-0.031607	0.015903	0.667388	1.000000	0.517037	0.523270	-0.221467
Cloud9am	0.077625	-0.289865	0.198195	-0.185032	-0.675610	0.071235	0.024280	0.052780	0.452182	0.517037	1.000000	0.604118	-0.137843
Cloud3pm	0.020489	-0.279053	0.171993	-0.184287	-0.704202	0.109088	0.053584	0.025269	0.358043	0.523270	0.604118	1.000000	-0.127869
Temp9am	0.901813	0.887020	0.011477	0.545497	0.291139	0.150258	0.129298	0.163601	-0.472826	-0.221467	-0.137843	-0.127869	1.000000
Temp3pm	0.708865	0.984562	-0.079178	0.574275	0.490180	0.032970	0.005108	0.028567	-0.499777	-0.557989	-0.302520	-0.318254	0.860574
RISK_MM	0.124743	-0.044208	0.308557	-0.043498	-0.294973	0.162923	0.069404	0.049240	0.172417	0.313183	0.198095	0.234814	0.051232

Si queremos ver la correlación entre variables cualitativas, podemos utilizar la tabla de contingencia, la cual nos muestra la relación que existe entre dos o más variables.

En el siguiente código, vemos la correlación que existe entre RainTomorrow y RainToday.

```
[ ] #Tabla de Contingencia
#Mide la relación entre las variables Cualitativas.

data_crosstab = pd.crosstab(df.RainTomorrow,
                            df.RainToday,
                            margins = True,
                            margins_name= 'Total',
                            normalize = True)
print(data_crosstab)

RainToday      No      Yes    Total
RainTomorrow
No            0.658640  0.119741  0.778382
Yes           0.117937  0.103681  0.221618
Total          0.776577  0.223423  1.000000
```

Con la función `Describe()` vemos la variedad de estadísticas para cada variable cuantitativa, entre ellas se encuentran:

- La media. (`mean`)
- Los cuartiles. (25%, 50%, 75%)
- Desviación típica estándar (`Std`)
- El valor mínimo (`min`)
- El valor máximo (`max`)
- La mediana. (50%, este cuartil se interpreta como la mediana)

```
[ ] #con la función describe() podemos ver un resumen de las principales estadísticas
#del dataset
df.describe()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm
count	141556.000000	141871.000000	140787.000000	81350.000000	74377.000000	132923.000000	140845.000000	139563.000000	140419.000000	138583.000000
mean	12.186400	23.226784	2.349974	5.469824	7.624853	39.984292	14.001988	18.637576	68.843810	51.482606
std	6.403283	7.117618	8.465173	4.188537	3.781525	13.588801	8.893337	8.803345	19.051293	20.797772
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000	0.000000
25%	7.600000	17.900000	0.000000	2.600000	4.900000	31.000000	7.000000	13.000000	57.000000	37.000000
50%	12.000000	22.600000	0.000000	4.800000	8.500000	39.000000	13.000000	19.000000	70.000000	52.000000
75%	16.800000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	83.000000	66.000000
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	100.000000	100.000000

<

Finalmente podemos configurar y desplegar la información de nuestro dataset en gráficos para poder visualizar nuestros datos de manera más amigable y personalizada al poder tomar ciertas columnas específicas para mostrar su relación:

Hallazgo 1:

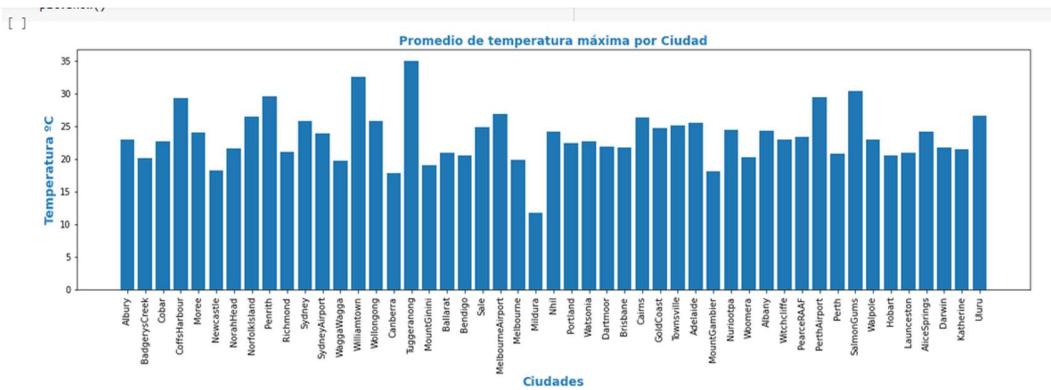
Promedio de temperatura máxima por Ciudad:

Como primer hallazgo, investigaremos cómo se ve de forma gráfica la distribución de temperatura máxima por ciudad.

Para esto agrupamos por ciudad y calculamos la media.

```
[ ] serie_MaxTemp_Location = df.groupby('Location')['MaxTemp'].mean()
list_location = df.loc[:, "Location"].unique()
```

```
[ ] #Exploración de los datos en forma Gráfica
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20, 5)
fig, ax = plt.subplots()
ax.bar(list_location, serie_MaxTemp_Location)
ax.set_title('Promedio de temperatura máxima por Ciudad', loc = "center", fontdict = {'fontsize':14, 'fontweight':'bold', 'color':'tab:blue'})
ax.set_xlabel("Ciudades", fontdict = {'fontsize':14, 'fontweight':'bold', 'color':'tab:blue'})
ax.set_ylabel("Temperatura °C", fontdict = {'fontsize':14, 'fontweight':'bold', 'color':'tab:blue'})
plt.xticks(rotation=90)
plt.show()
```



Podemos observar la media que tiene cada ciudad en temperatura máxima y vemos que 'Mildura' posee la temperatura máxima promedio más baja mientras que, en su contraparte, 'Tuggeranong' tiene la más alta.

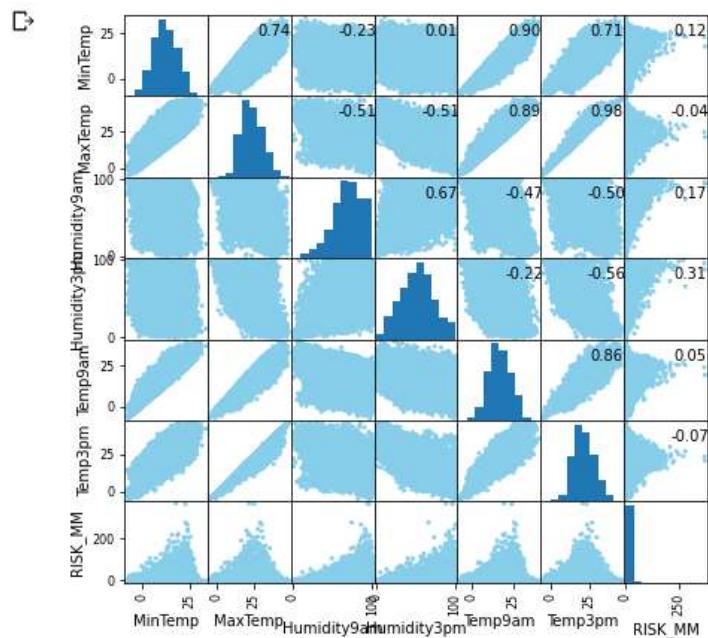
Hallazgo 2:

Gráficos sobre la correlación de Pearson:

El siguiente gráfico muestra la correlación de forma gráfica, donde podemos ver la correlación de las características con respecto a MaxTemp (Característica a utilizar más adelante para predecir)

```
▶ from pandas.plotting import scatter_matrix
import numpy as np
import matplotlib.pyplot as plt

axes = scatter_matrix(df_2, alpha=1, figsize=(7,7), color="skyblue")
corr = df_2.corr().values
for i, j in zip(*plt.np.triu_indices_from(axes, k=1)):
    axes[i, j].annotate("%.2f" %corr[i,j], (0.8, 0.8), xycoords='axes fraction', ha='center', va='center')
plt.show()
```



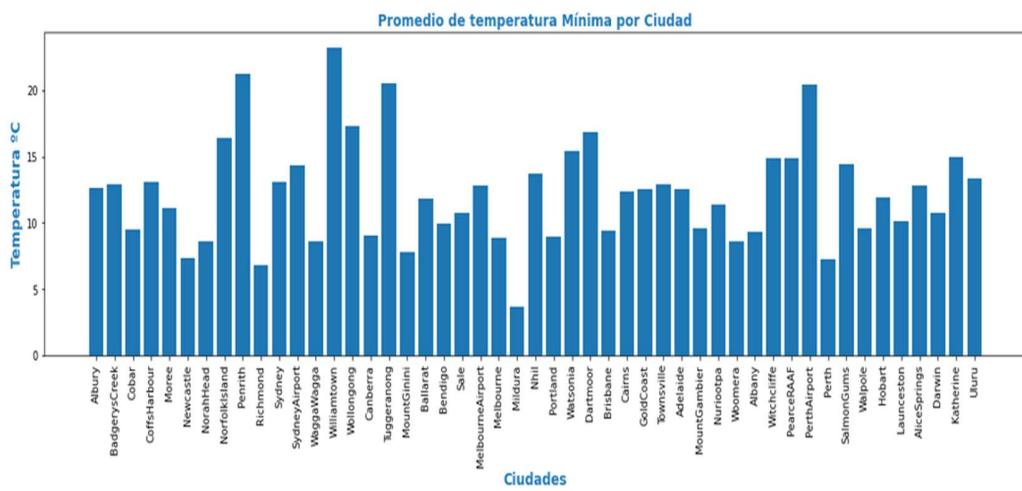
Hallazgo 3:

Promedio de Velocidad del viento por Ciudad:

Veremos la distribución que tiene la velocidad del viento por ciudad. Recordemos que un aspecto importante además de la temperatura es la velocidad del viento. Para poder ver la distribución, calculamos la media por ciudad.

```
[ ] serie_wind_speed_Location = df.groupby('Location')[['WindGustSpeed']].mean()

❶ plt.rcParams["figure.figsize"] = (20, 5)
fig, ax = plt.subplots()
ax.bar(list_location, serie_wind_speed_Location)
ax.set_title('Promedio de velocidad del viento por Ciudad', loc = "center", fontdict = {'fontsize':14, 'fontweight':'bold', 'color':'tab:blue'})
ax.set_xlabel('Ciudades', fontdict = {'fontsize':14, 'fontweight':'bold', 'color':'tab:blue'})
ax.set_ylabel("Velocidad del viento (km/hr)", fontdict = {'fontsize':14, 'fontweight':'bold', 'color':'tab:blue'})
plt.xticks(rotation=90)
plt.show()
```



Limpieza de valores nulos

En primer lugar, ejecutaremos un par de funciones que nos permitirán filtrar e identificar los valores nulos existentes en cada columna:

```
▶ # La siguiente porción de código nos permite filtrar el data set
# Nos entrega la cantidad de nulos que tiene cada Característica y la ordena del mayor a menor.
list_nulls = df.isnull().sum()
list_nulls[(list_nulls>0)].sort_values(ascending = False)

▶ Sunshine      67816
Evaporation    60843
Cloud3pm        57094
Cloud9am         53657
Pressure9am     14014
Pressure3pm     13981
WindDir9am      10013
WindGustDir     9330
WindGustSpeed   9270
WindDir3pm       3778
Humidity3pm     3610
Temp3pm          2726
WindSpeed3pm    2630
Humidity9am     1774
RainToday        1406
Rainfall         1406
WindSpeed9am    1348
Temp9am          904
MinTemp          637
MaxTemp          322
dtype: int64
```

A continuación, filtramos los datos con la media de cada ciudad, ya que cada ciudad tiene su media y no es la misma en los estados del norte donde hay estaciones de lluvia y sequía (2 estaciones) que la del resto del país que tienen las 4 estaciones.

```
[25] means = round(df.groupby('Location').transform(np.mean),2)
df = df.fillna(means)
```

Eliminamos las columnas con mayor cantidad de nulos, porque consideramos que si se imputa con la media, se podría sesgar el modelo. y eliminamos la fecha ya que no representa un dato importante para nuestro caso.

```
[27] del df['Sunshine']
      del df['Evaporation']
      del df['Date']
```

En este caso no utilizaremos las columnas “Sunshine”, “Evaporation” y “Date”, si bien es cierto que la cantidad de horas de sol (Sunshine) es importante en nuestro caso. La cantidad de datos nulos son muy altas como para reemplazarlo.

Como análisis del problema, determinamos que no debemos reemplazar el valor de RainToday ya que, queremos mantener la calidad de los datos.

Por esto, eliminamos las filas (registros) que sean nulos en RainToday.

Calculamos la moda para las variables ‘WindGustDir’, ‘WindDir9am’ y ‘WindDir3pm’.

Luego calculamos la media teniendo en cuenta todo el dataset para las variables donde no hay un promedio por ciudad.

Por último, eliminamos las columnas restantes que queden con Valores nulos, las cuales son ‘Pressure9am’ y ‘Pressure3pm’

```
[510] df = df.dropna(subset = ['RainToday'])
      df['WindGustDir'] = df['WindGustDir'].fillna(df['WindGustDir'].mode()[0])
      df['WindDir9am'] = df['WindDir9am'].fillna(df['WindDir9am'].mode()[0])
      df['WindDir3pm'] = df['WindDir3pm'].fillna(df['WindDir3pm'].mode()[0])
      df.fillna(df.mean(), inplace=True)
      df = df.dropna(axis = 1)
```

Como podemos apreciar en el despliegue, ya no aparecen datos nulos en ninguna columna.

```
▶ list_nulls = df.isnull().sum()
    list_nulls[(list_nulls>0)].sort_values(ascending = False)
    ▶ Series([], dtype: int64)
```

A continuación, vemos cómo quedó el dataset (df) luego de la imputación de datos nulos.

▶ df.info()

```
↳ <class 'pandas.core.frame.DataFrame'>
Int64Index: 140787 entries, 0 to 142192
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Location          140787 non-null   object  
 1   MinTemp            140787 non-null   float64 
 2   MaxTemp            140787 non-null   float64 
 3   Rainfall           140787 non-null   float64 
 4   WindGustDir        140787 non-null   object  
 5   WindGustSpeed      140787 non-null   float64 
 6   WindDir9am         140787 non-null   object  
 7   WindDir3pm         140787 non-null   object  
 8   WindSpeed9am       140787 non-null   float64 
 9   WindSpeed3pm       140787 non-null   float64 
 10  Humidity9am        140787 non-null   float64 
 11  Humidity3pm        140787 non-null   float64 
 12  Cloud9am           140787 non-null   float64 
 13  Cloud3pm           140787 non-null   float64 
 14  Temp9am            140787 non-null   float64 
 15  Temp3pm             140787 non-null   float64 
 16  RainToday           140787 non-null   object  
 17  RISK_MM             140787 non-null   float64 
 18  RainTomorrow        140787 non-null   object  
dtypes: float64(13), object(6)
memory usage: 21.5+ MB
```

Como vemos, apenas hubo una pérdida de datos la cual no es significativa.

Estandarización de datos

Para la estandarización de los datos utilizaremos 3 técnicas:

- Standard Scaler
- Binary Encoder
- OneHot Encoder

Standard Scaler estandariza los datos eliminando la media y escalando los datos de forma que su varianza sea igual a 1, esto solo sirve para las variables cuantitativas.

Para las variables cualitativas, usaremos One Hot Encoder para la variable 'RainToday' la cual separa en las opciones que tenga la columna, en este caso 'Si' y 'No'. Mientras el resto de las variables Cualitativas, realizamos una estandarización con Binary Encoder() el cual sirve para el caso en que los registros sean muchos tipos, los convierte en binario; disminuyendo la cantidad creadas en comparación a OneHot en los mismos casos.

Primero Instalamos la libreria Category_Encoders la cual contiene entre muchas funciones, la que utilizaremos, BinaryEncoder()

```
▶ pip install category_encoders
Collecting category_encoders
  Downloading category_encoders-2.4.0-py2.py3-none-any.whl (86 kB)
    ██████████ | 86 kB 3.7 MB/s
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (0.5.2)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.0.2)
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.3.5)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.4.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (0.10.2)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.21.6)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>0.21.1->category_encoders) (2022.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>0.21.1->category_encoders) (2.8.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy>=0.5.1->category_encoders) (1.15.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->category_encoders) (3.1.0)
Installing collected packages: category-encoders
Successfully installed category-encoders-2.4.0
```

Separamos el Set de datos en Train y test con un tamaño 80% y 20% respectivamente y retiramos las columnas objetivo.

```
[32] from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(
          df.drop(['RainTomorrow', 'RISK_MM'], axis = 'columns'),
          df['RainTomorrow'],
          train_size = 0.8,
          random_state = 1234,
          shuffle = True
      )
```

Separamos las columnas que se transformarán en una variable representativa.

```
[521] columnas_numeric = X_train.select_dtypes(include=['float64']).columns.to_list()
      columnas_binary = []
      columnas_binary.append(X_train.columns.values[0])
      columnas_binary.append(X_train.columns.values[4])
      columnas_binary.append(X_train.columns.values[6])
      columnas_binary.append(X_train.columns.values[7])

      columnas_onehot = []
      columnas_onehot.append(X_train.columns.values[16])
```

Importamos las librerías necesarias para estandarizar de SkLearn, la cual es Vital para el manejo de datos y machine learning.

Utilizamos una función que permite estandarizar las columnas con distintos tipos de funciones.

```
[35] from sklearn.preprocessing import StandardScaler
     from sklearn.preprocessing import OneHotEncoder
     from sklearn.compose import ColumnTransformer
     from category_encoders import BinaryEncoder

     preprocessor = ColumnTransformer(
         [('scale', StandardScaler(), columnas_numeric),
          ('onehot', OneHotEncoder(), columnas_Rain_Today),
          ('binary', BinaryEncoder(), columnas_Location)],
         remainder='passthrough')
```

Hacemos la transformación de las columnas.

```
[36] X_train_prep = preprocessor.fit_transform(X_train)
     X_test_prep = preprocessor.transform(X_test)
```

Recuperamos el nombre de las columnas y las ingresamos nuevamente a un dataframe.

```
[37] decodificar_nombre = preprocessor.named_transformers_['onehot'].get_feature_names(columnas_Rain_Today)
     nombre_columnas = np.concatenate([columnas_numeric, decodificar_nombre, preprocessor.named_transformers_["binary"].get_feature_names()])
     Datos_train_prep = preprocessor.transform(X_train)
     df = pd.DataFrame(Datos_train_prep, columns=nombre_columnas)
     df.info()
```

La siguiente imagen muestra como el tipo de dato cambió a Float en todos los casos.

Para la computadora es más rápido y eficiente trabajar con números que con tipos object. también hacemos que ningún dato tenga más importancia que otro.

Data columns (total 35 columns):			
#	Column	Non-Null Count	Dtype
0	MinTemp	112629 non-null	float64
1	MaxTemp	112629 non-null	float64
2	Rainfall	112629 non-null	float64
3	WindGustSpeed	112629 non-null	float64
4	WindSpeed9am	112629 non-null	float64
5	WindSpeed3pm	112629 non-null	float64
6	Humidity9am	112629 non-null	float64
7	Humidity3pm	112629 non-null	float64
8	Cloud9am	112629 non-null	float64
9	Cloud3pm	112629 non-null	float64
10	Temp9am	112629 non-null	float64
11	Temp3pm	112629 non-null	float64
12	RainToday_No	112629 non-null	float64
13	RainToday_Yes	112629 non-null	float64
14	Location_0	112629 non-null	float64
15	Location_1	112629 non-null	float64
16	Location_2	112629 non-null	float64
17	Location_3	112629 non-null	float64
18	Location_4	112629 non-null	float64
19	Location_5	112629 non-null	float64
20	WindGustDir_0	112629 non-null	float64
21	WindGustDir_1	112629 non-null	float64
22	WindGustDir_2	112629 non-null	float64
23	WindGustDir_3	112629 non-null	float64
24	WindGustDir_4	112629 non-null	float64
25	WindDir9am_0	112629 non-null	float64
26	WindDir9am_1	112629 non-null	float64
27	WindDir9am_2	112629 non-null	float64
28	WindDir9am_3	112629 non-null	float64
29	WindDir9am_4	112629 non-null	float64
30	WindDir3pm_0	112629 non-null	float64
31	WindDir3pm_1	112629 non-null	float64
32	WindDir3pm_2	112629 non-null	float64
33	WindDir3pm_3	112629 non-null	float64
34	WindDir3pm_4	112629 non-null	float64
dtypes: float64(35)			

Realizar razonamiento y justificación respecto del análisis de los datos

La variable objetivo (Target) para los modelos de regresión será “MaxTemp” (Temperatura mínima).

La variable objetivo (Target) para los modelos de clasificación será “RainTomorrow” (llueve mañana).

Selección de modelos a utilizar

La selección de modelos se realizará mediante las mejores métricas de desempeño según su categoría.

Los modelos de clasificación a utilizar serán los siguientes:

- RandomForestClassifier (Clasificador de bosques aleatorios)
- SVC (Clasificador maquina de soporte de vectores)
- K-NeighborsClassifier (Clasificador de K vecinos cercanos)

De los cuales se medirán las siguientes métricas:

- F1-Score
- Recall
- Accuracy
- Precision
- ROC AUC (Área sobre la curva)

Los modelos de Regresión a utilizar son los siguientes:

- LinearRegression (Regresión Lineal Multiple)
- SVR (Regresor Maquina de soporte de vectores)
- GradientBoostingRegressor
- RandomForestRegressor
- KNeighborsRegressor (Regresor K-Vecinos cercanos)

Los cuales se medirán las siguientes métricas:

- Error cuadrático medio (MSE).
- Raíz error cuadrático medio (RMSE).
- Error absoluto medio (MAE)
- R² Score

Implementación de modelos

Para la correcta implementación de los modelos, sepáramos el dataSet en Train y Test en una proporción de 80-20 respectivamente.

Modelos de Clasificación

```
[ ] #Split para el target RainTomorrow  
  
X_train, X_test, y_train, y_test = train_test_split(  
    df,  
    target_Tomorrow,  
    train_size    = 0.8,  
    random_state = 1234,  
    shuffle      = True  
)
```

Antes de comenzar con la implementación de modelos, debemos Importar las librerías respectivas de cada modelo.

```
#modelos de Clasificación
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

Para la visualización de las métricas, utilizaremos una función de creación propia donde mostramos varios datos al mismo tiempo, con el uso de varias librerías de Sklearn que se importan previamente. Se visualiza una matriz de confusión y un reporte de métricas importantes.

```
#Metricas para Clasificación
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
```

```
[ ] LABELS = ["0","1"]
def mostrar_resultados(y_test, pred_y):
    conf_matrix = confusion_matrix(y_test, pred_y)
    plt.figure(figsize=(8, 8))
    sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
    plt.title("Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()
    print (classification_report(y_test, pred_y))
```

Random Forest Classifier

Uno de los modelos más utilizados en el rubro para problemas de clasificación, esto se debe a que una de las características del modelo es que puede funcionar con “missing values”. Como inconveniente encontramos que puede caer en Overfitting

si no se le coloca un límite en la profundidad de los árboles. En nuestro caso solo ajustamos la cantidad de árboles a generar que por defecto son 100; colocamos 1000.

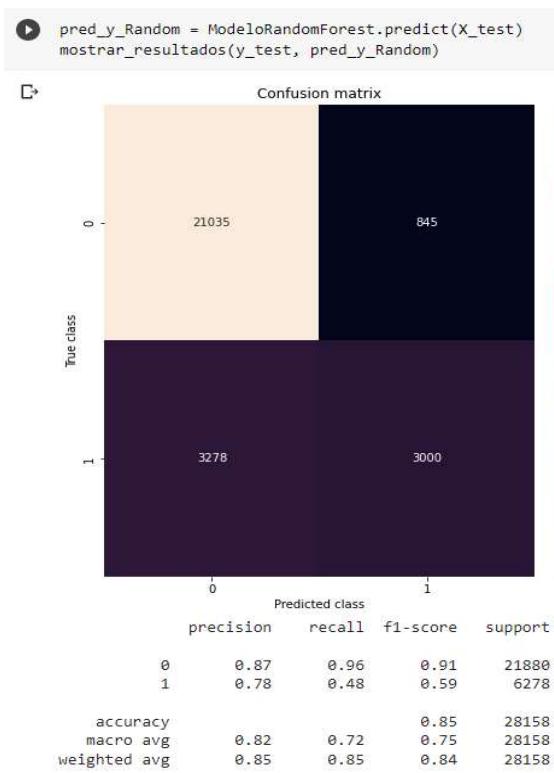
Primero, cargamos el modelo en una variable y realizamos lo que se conoce como 'fit' para ejecutar o implementar el modelo en el set de datos.

```
[ ] #modelo RandomForestClassifier  
ModeloRandomForest = RandomForestClassifier(n_estimators = 1000)  
ModeloRandomForest.fit(X_train,y_train)  
print(ModeloRandomForest.score(X_test,y_test))
```

```
0.8535762483130904
```

El Score dado por el modelo es de: 0.8535.

El Score es la media del accuracy dado por el set de pruebas.



En la matriz de confusión podemos ver la cantidad de verdaderos positivos del modelo (21035) y la cantidad de verdaderos negativos (3000) son la cantidad de aciertos que tuvo el modelo respecto al set de prueba.

También vemos los falsos positivos (845) y los falsos negativos (3278) son la cantidad de veces que el modelo se equivocó en la predicción.

Vemos que el modelo es mejor al predecir una clase que la otra.

SVC

“SVC ha resultado ser uno de los mejores clasificadores para un amplio abanico de situaciones, por lo que se considera uno de los referentes dentro del ámbito de aprendizaje estadístico y *machine learning*.”

Referencia: <https://www.cienciadedatos.net/documentos/py24-svm-python.html>

El modelo SVC tiene hiperparametros interesantes para modificar tales como el tipo de función que utilizaremos o el peso de las clases en caso de que exista mucha diferencia en la cantidad de muestra por clases.

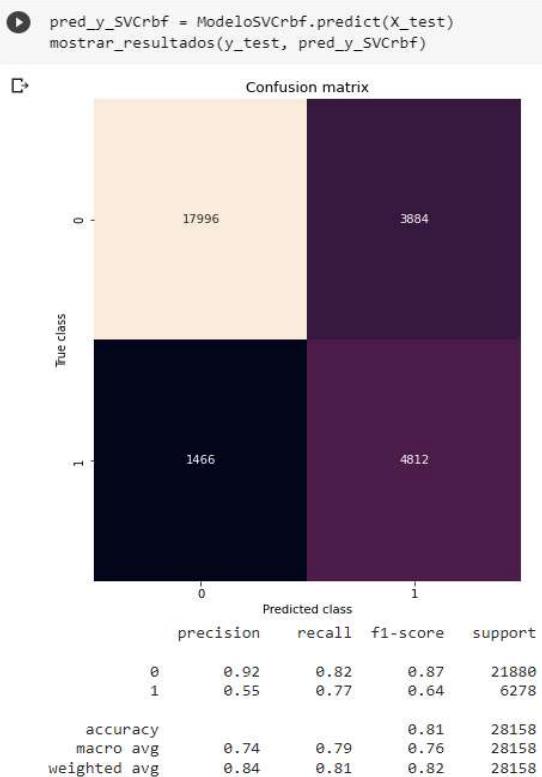
Primero, cargamos el modelo en una variable y realizamos lo que se conoce como ‘fit’ para ejecutar o implementar el modelo en el set de datos.

```
[ ] #modelo SVC rbf
ModeloSVCrbf = SVC(kernel='rbf', C=1, class_weight="balanced")
ModeloSVCrbf.fit(X_train,y_train)
print(ModeloSVCrbf.score(X_test,y_test))

0.8100007102777186
```

Modificamos el hiperparametro ‘Class_weight’ a Balanced, esto hace que el peso de las variables sea iguales o parecidos y ninguno tenga más importancia que otro.

El Score dado por el modelo es de: 0.8100.



En la matriz de confusión podemos ver la cantidad de verdaderos positivos del modelo (17996) y la cantidad de verdaderos negativos (4812) son la cantidad de aciertos que tuvo el modelo respecto al set de prueba.

También vemos los falsos positivos (3884) y los falsos negativos (1466) son la cantidad de veces que el modelo se equivocó en la predicción.

Vemos que el modelo es mejor al predecir una clase que la otra.

K-Neighbors Classifier

Sirve esencialmente para clasificar valores buscando los puntos de datos “más similares” (por cercanía) aprendidos en la etapa de entrenamiento y haciendo conjeturas de nuevos puntos basado en esa clasificación.

Primero, cargamos el modelo en una variable y realizamos lo que se conoce como ‘fit’ para ejecutar o implementar el modelo en el set de datos.

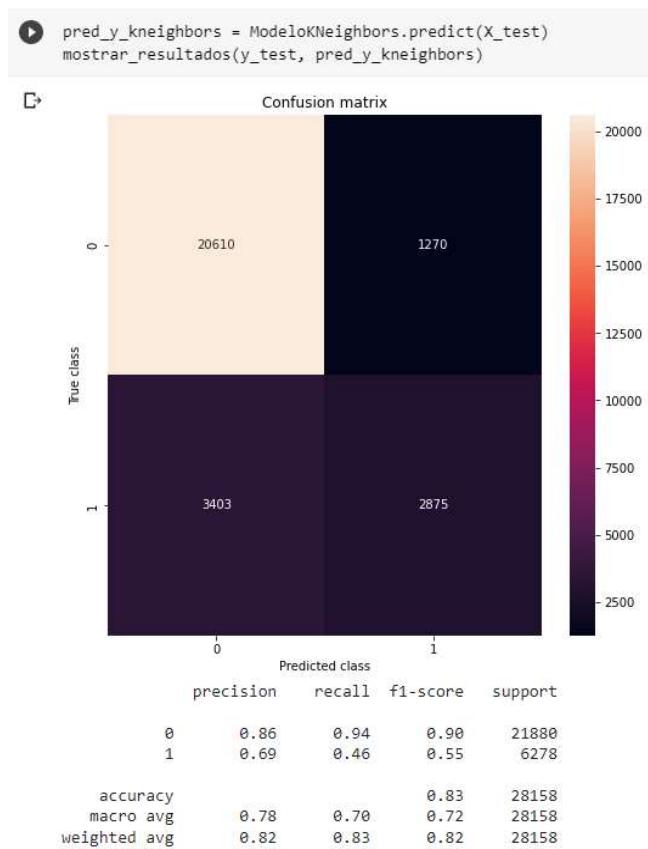
```

[ ] #modelo KNeighborsClassifier
ModeloKNeighbors = KNeighborsClassifier()
ModeloKNeighbors.fit(X_train,y_train)
print(ModeloKNeighbors.score(X_test,y_test))

```

0.8340436110519213

El Score dado por el modelo es de: 0.8340.



En la matriz de confusión podemos ver la cantidad de verdaderos positivos del modelo (20610) y la cantidad de verdaderos negativos (2875) son la cantidad de aciertos que tuvo el modelo respecto al set de prueba.

También vemos los falsos positivos (1270) y los falsos negativos (3403) son la cantidad de veces que el modelo se equivocó en la predicción.

Vemos que el modelo es mejor al predecir una clase que la otra.

Modelos de Regresión

Para los modelos de regresión, acotaremos el set de datos a las características con mayor correlación con el target MaxTemp (La correlación está dada en la matriz de correlación de Pearson).

```
[111] df = df[['MinTemp', 'Temp9am', 'Temp3pm']]  
      #Split para el target RISK_MM  
  
      X_train, X_test, y_train, y_test = train_test_split(  
          df,  
          target_maxtemp,  
          train_size    = 0.8,  
          random_state = 1234,  
          shuffle      = True  
      )
```

Importamos las librerías necesarias para poder implementar los modelos de regresión y visualización de métricas dichas anteriormente.

```
#Modelos de Regresión
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
#Metricas para la regresión
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from math import sqrt
```

Para la visualización de las métricas, utilizaremos una función de creación propia donde mostramos varios datos al mismo tiempo, con el uso de varias librerías de Sklearn que se importan previamente.

```
[118] def metricas_regresion(y_test, y_tongo):
    print("Error medio cuadrado      :", mean_squared_error(y_test, y_tongo))
    print("Raiz de Error medio cuadrado : ", sqrt(mean_squared_error(y_test, y_tongo)))
    print("R² Cuadrado                 :", r2_score(y_test, y_tongo))
    print("Error Medio Absoluto        :", mean_absolute_error(y_test, y_tongo))
```

Linear Regresión

Primero, cargamos el modelo en una variable y realizamos lo que se conoce como ‘fit’ para ejecutar o implementar el modelo en el set de datos.

```
[122] modelo_linear_regressor = LinearRegression()
      modelo_linear_regressor.fit(X_train,y_train)

      LinearRegression()
```

Hacemos uso de la función creada previamente para visualizar las métricas las cuales serán evaluadas en la siguiente etapa.

```
[123] y_tongo_linear_regressor = modelo_linear_regressor.predict(X_test)
      metricas_regresion(y_test,y_tongo_linear_regressor)

      Error medio cuadrado      : 1.516835631453535
      Raiz de Error medio cuadrado : 1.2315988110799454
      R2 Cuadrado              : 0.9700369750781845
      Error Medio Absoluto       : 0.8229113875870372
```

SVR

Primero, cargamos el modelo en una variable y realizamos lo que se conoce como ‘fit’ para ejecutar o implementar el modelo en el set de datos.

```
[128] modelo_SVR = SVR()
      modelo_SVR.fit(X_train,y_train)

      SVR()
```

Hacemos uso de la función creada previamente para visualizar las métricas las cuales serán evaluadas en la siguiente etapa.

```
[129] y_tongo_SVR = modelo_SVR.predict(X_test)
      metricas_regresion(y_test,y_tongo_SVR)

      Error medio cuadrado      : 1.244504896679347
      Raiz de Error medio cuadrado : 1.11557379705663
      R2 Cuadrado              : 0.9754164983592905
      Error Medio Absoluto       : 0.7024778191747797
```

GradientBoostingRegressor

Primero, cargamos el modelo en una variable y realizamos lo que se conoce como ‘fit’ para ejecutar o implementar el modelo en el set de datos.

```
[126] modelo_GradientBoostingRegressor = GradientBoostingRegressor()
      modelo_GradientBoostingRegressor.fit(X_train,y_train)

      GradientBoostingRegressor()
```

Hacemos uso de la función creada previamente para visualizar las métricas las cuales serán evaluadas en la siguiente etapa.

```
[127] y_tongo_GradientBoostingRegressor = modelo_GradientBoostingRegressor.predict(X_test)
      metricas_regresion(y_test,y_tongo_GradientBoostingRegressor)

      Error medio cuadrado      : 1.2069403818962319
      Raiz de Error medio cuadrado : 1.0986083842280796
      R2 Cuadrado              : 0.9761585342590826
      Error Medio Absoluto       : 0.732093770829632
```

Random Forest Regressor

Primero, cargamos el modelo en una variable y realizamos lo que se conoce como ‘fit’ para ejecutar o implementar el modelo en el set de datos.

```
[124] modelo_random_forest_regressor = RandomForestRegressor()
      modelo_random_forest_regressor.fit(X_train,y_train)

      RandomForestRegressor()
```

Hacemos uso de la función creada previamente para visualizar las métricas las cuales serán evaluadas en la siguiente etapa.



```
y_tongo_RandomForestRegressor = modelo_random_forest_regressor.predict(X_test)
metricas_regresion(y_test,y_tongo_RandomForestRegressor)

Error medio cuadrado      : 1.2342586182830861
Raiz de Error medio cuadrado : 1.1109719250652044
R2 Cuadrado              : 0.9756188996535222
Error Medio Absoluto       : 0.7648325423742137
```

K-NeighborsRegressor

Primero, cargamos el modelo en una variable y realizamos lo que se conoce como ‘fit’ para ejecutar o implementar el modelo en el set de datos

```
[130] modelo_kneighbors_regressor = KNeighborsRegressor()
      modelo_kneighbors_regressor.fit(X_train,y_train)

      KNeighborsRegressor()
```

Hacemos uso de la función creada previamente para visualizar las métricas las cuales serán evaluadas en la siguiente etapa.

```
[131] y_tongo_KNeighborsRegressor = modelo_kneighbors_regressor.predict(X_test)
      metricas_regresion(y_test,y_tongo_KNeighborsRegressor)
```

```
Error medio cuadrado      : 1.3336338593650117
Raiz de Error medio cuadrado : 1.154830662636307
R2 Cuadrado              : 0.9736558769215893
Error Medio Absoluto       : 0.7832885148092904
```

Evaluación de modelos

Para los modelos de clasificación tenemos la siguiente tabla:

Modelo	Accuracy	F1-score avg	Recall avg	Precision avg
Random Forest Classifier	0.85	0.75	0.72	0.82
SVC	0.81	0.76	0.79	0.74
K-Neighbors Classifier	0.83	0.72	0.70	0.78

El modelo que seleccionaremos será SVC ya que, si bien es cierto que no tiene el mejor accuracy, es el que mejor predice cuando llueve al día siguiente. Es el que tiene las mejores métricas al momento de predecir el “Sí” en comparación con los otros modelos. Esto lo determinamos por lo solicitado por la empresa, ya que necesitan saber con precisión si los días siguientes lloverá o no.

Es momento de evaluar las métricas para los modelos de regresión, tenemos la siguiente tabla que muestra las métricas:

Modelos	MSE	RMSE	R ² Score	MAE
LinearRegresion	1.52	1.24	0.97	0.82
SVR	1.24	1.12	0.98	0.70
GradientBoostingRegressor	1.21	1.10	0.98	0.73
RandomForestRegressor	1.24	1.11	0.98	0.76
K-NeighborsRegressor	1.33	1.15	0.97	0.78

Teniendo en cuenta esto datos, Hacemos uso del modelo SVR ya que es el que tiene las mejores métricas en promedio. Determinamos que es el modelo que presenta la mejor predicción para nuestra problemática. Si bien es cierto que el modelo GradientBoostingRegressor tiene mejores métricas, el tiempo de implementación y modelado es mucho más extenso que el resto de los modelos probados. A pesar de esto, todos los modelos que fueron testeados tuvieron buenas métricas.

Fases

FASE 1: Business Understanding	Determinación de los objetivos. Definición de criterios de éxito. Calificación de la situación. Determinación de las metas de la minería de datos.
FASE 2: Data Understanding	Recopilación de datos iniciales. Descripción de los datos. Exploración de los datos. Verificación de la Calidad de los datos,
FASE 3: Data Preparation	Selección de Datos. Limpieza de los Datos. Construcción de nuevos Datos. Formateo de los Datos.
FASE 4: Modeling	Selección de la Técnica de Modelación. Generación de pruebas para el Modelo. Construcción del Modelo. Calificación del Modelo.
FASE 5: Evaluation	Evaluación del modelo con respecto a los objetivos del proyecto. Evaluación costos-beneficio. Evaluar su aplicación en la realidad.
FASE 6: Deployment	Aplicación del modelo a la rutina diaria. Monitoreo y Mantenimiento. Reportes finales.

Toma de decisiones

Generación de CSV

Para posteriormente crear los gráficos necesarios que permitirán mostrar los resultados obtenidos, debemos primero exportar el archivo .CSV que nos dio el Jupyter Notebook.

La generación del .CSV se realiza con la siguiente porción de código.

```
[ ] df.to_csv('australia.csv', sep=';')
```

La función de dataframe ‘to_csv’ permite convertir en archivo de Excel csv el dataframe. Hay que proporcionar el nombre que le queremos dar al archivo, y como queremos que sea separado (ya que al ser un archivo csv se puede separar por ‘coma’, ‘punto y coma’ o ‘tabulaciones’).

Gráficos con POWER BI

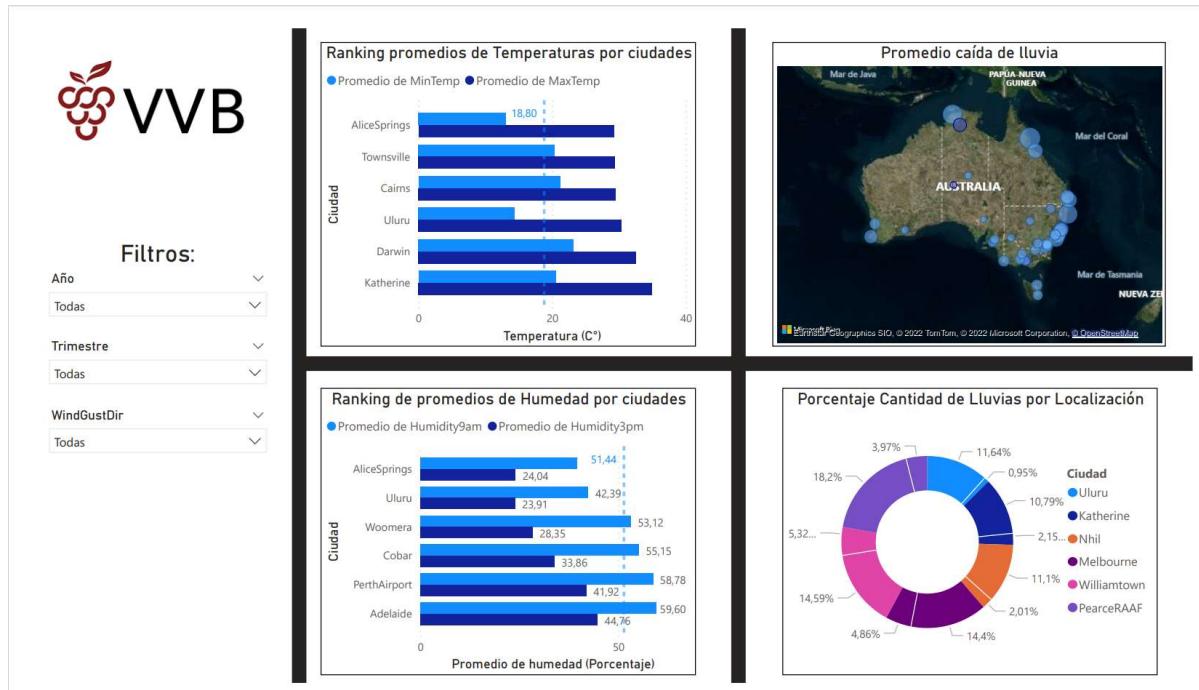
Los gráficos generados a partir del archivo .CSV se dividen en dos importantes partes:

- Gráficos Interactivos (Propuesta general)
- Gráficos estáticos (Nuestra recomendación)

Fueron creados con la aplicación Power BI, la cual es la más utilizada en el mercado actualmente.

Gráficos Interactivos

La siguiente imagen nos muestra una vista general de los gráficos.



Los gráficos son interactivos porque se pueden ver diferentes etapas según sea necesario (a gusto del cliente). Del lado izquierdo podemos ver los filtros que podemos aplicar:

- Por año
- Por Trimestre
- Por dirección del viento

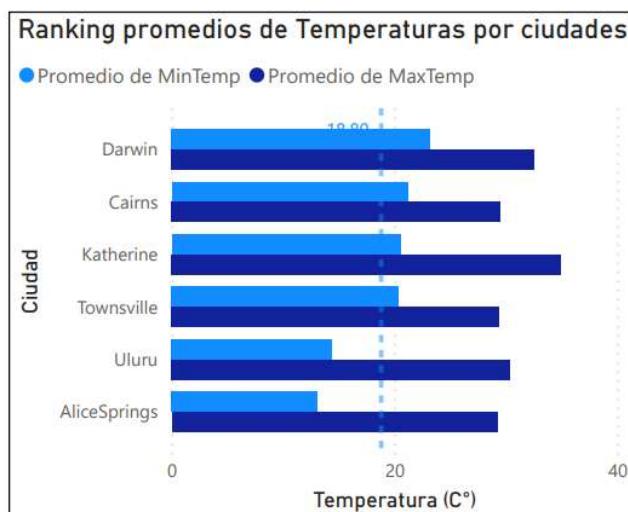
La empresa VVB nos encomendó la tarea de encontrar una de las mejores zonas para poder instalar un viñedo. Por lo que investigamos las condiciones perfectas en la que se deben plantar las uvas.

Clima: Ya ubicados en la latitud adecuada, se deben descartar todas las regiones demasiado húmedas, de clima lluvioso y poca insolación. Las mejores son las regiones de clima seco y árido, con escasa lluvias anuales que se concentren sobre todo a fines del otoño y en el invierno. Otro factor importante, para la calidad de los viñedos, es la amplitud térmica, abundante sol durante el día y fresco durante la noche. Esta variación de temperatura facilita la formación de las sustancias aromáticas y ayuda a la fijación de pigmentos responsables del color y el desarrollo de los componentes que otorgan cuerpo y estructura al vino. Los vinos resultantes son de grandes cualidades aromáticas, de buen color y mucho cuerpo, condiciones imprescindibles para que un vino sea “Añejo”.

Altitud: Este factor cobra importancia especialmente en Argentina, donde la mayoría de los viñedos se encuentran entre los 350 y 2300 m. sobre el nivel del mar. A mayor altitud aumenta la amplitud térmica y mejora la radiación solar, favoreciendo esa maduración lenta de la que ya hablamos. Las zonas más altas se encuentran alejadas de las grandes poblaciones y por ende de la contaminación produciendo una materia prima especialmente sana.

Con esto en consideración, generamos los gráficos en consecuencia:

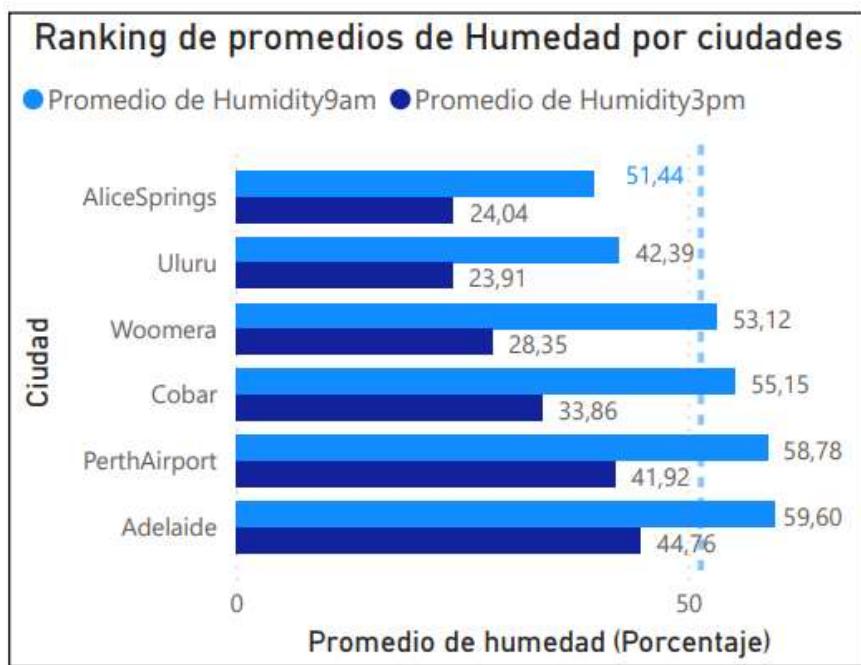
Ranking de temperaturas por ciudad



El gráfico nos muestra un ranking con los promedios de temperaturas por ciudades, ordenados de forma descendente por la temperatura mínima, filtrado por los 6 con mayor promedio de temperatura máxima.

Nos permite ver los que entran dentro del rango perfecto para el cultivo de vid (Entre 9C° y 18C°) y la maduración (Entre 18C° y 24C°). Se muestra una línea punteada que indica la media de la temperatura mínima de todas las ciudades.

Ranking de Humedad por ciudad



Los valores de humedad que necesita un viñedo dependen de la cepa a plantar. En norma general, lo recomendado es que oscile entre 25% y 65%.

El gráfico vemos el ranking de los promedios de humedad por ciudades, ordenados y filtrados por el promedio de humedad a las 9AM, se muestran solo los 6 primeros. Esto nos permite ver de igual forma cual será la manera óptima de regar el viñedo. Se ve una punteada que indica el promedio de la humedad de todas las ciudades a las 9AM.

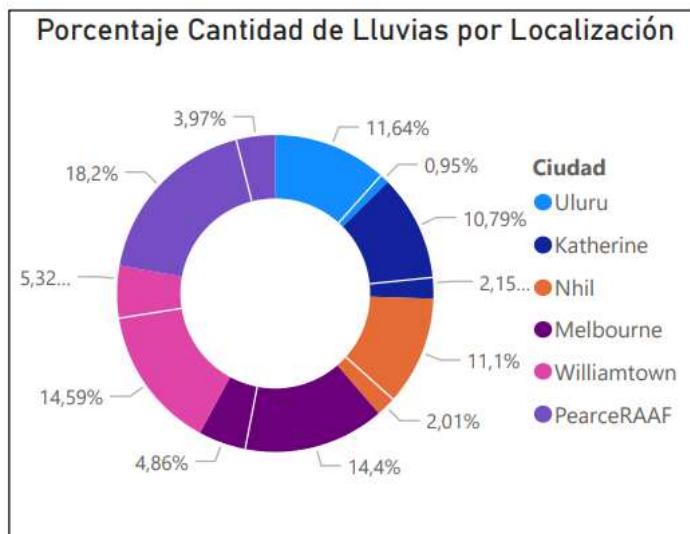
Mapa del promedio de caída de lluvia en Australia



La lluvia en exceso puede dañar la plantación, pero en su medida justa, permite ahorrar agua en el riego.

El Mapa nos muestra en formato de círculos, la cantidad de lluvia que cae en las distintas ciudades. Es un mapa interactivo en el cual se pueden ver a discreción los valores de cada ciudad (Ver Archivo POWER BI). Mientras más pequeño el circulo, menor es la cantidad de lluvia que cae en ese lugar.

Gráfico del porcentaje cantidad de lluvias por ciudad.



Se puede observar el porcentaje (Respecto del total) de la cantidad de veces que llueve por ciudad. Se encuentra ordenado de forma descendente ya que, queremos ver las ciudades en que llueve menos.

El gráfico nos muestra dos porcentajes por parte: El porcentaje mayor corresponde a la cantidad de veces que NO llovió y el porcentaje menor a la cantidad de veces que SI llovió.

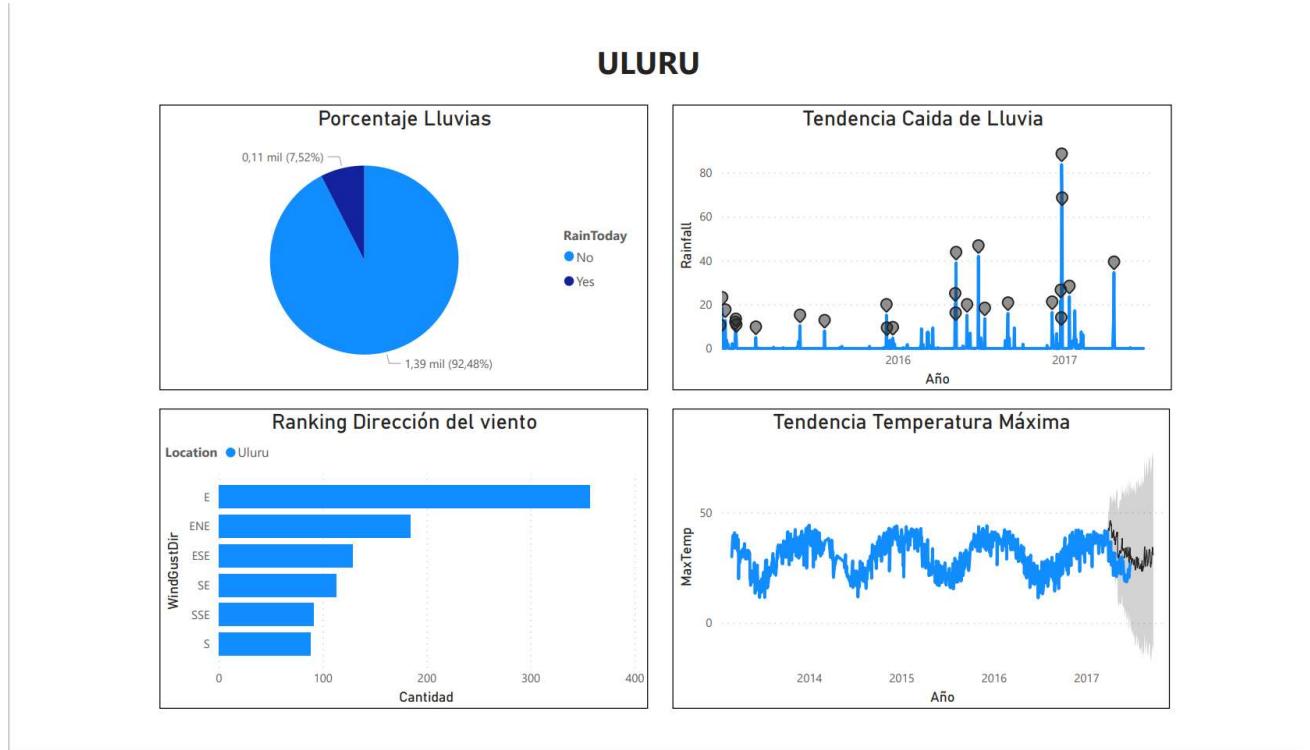
Vimos los gráficos interactivos que nos muestran una vista general de datos que cumplen con los requerimientos. Por lo que a continuación, se realiza un análisis de nuestra ciudad recomendada para instalar el viñedo.

Gráficos Estáticos

La ciudad que seleccionamos por los datos que disponemos es ULURU.

A continuación, vemos los gráficos que afirman y apoyan nuestra recomendación.

De esta misma forma, analizaremos cada uno con detalle.



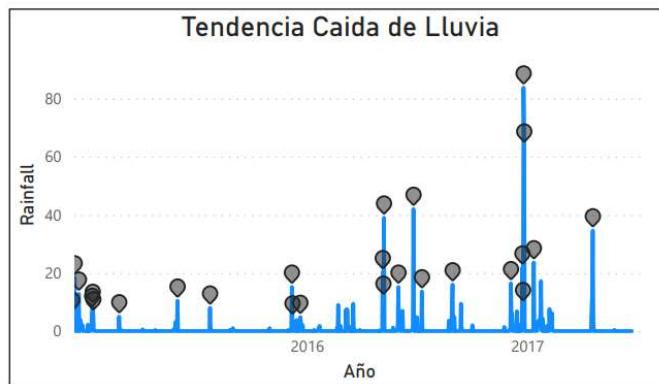
Grafica del porcentaje de Lluvias

El siguiente gráfico circular nos muestra la distribución de lluvias que tiene los datos, en un 92.48% no llueve y en un 7.52% llueve. Vemos que la cantidad de veces que llueve es significativamente menor a la cantidad de veces que no llueve. Esto nos indica que, la ciudad es principalmente no lluviosa.



Tendencia Caída de lluvia

Observamos la tendencia que tiene la caída de lluvia por año. Se marcan los puntos que tienen una varianza diferente a lo convencional. Vemos que hay aumentos notables al año 2017, se pudo deber a algún fenómeno meteorológico, pero no es la regla general.



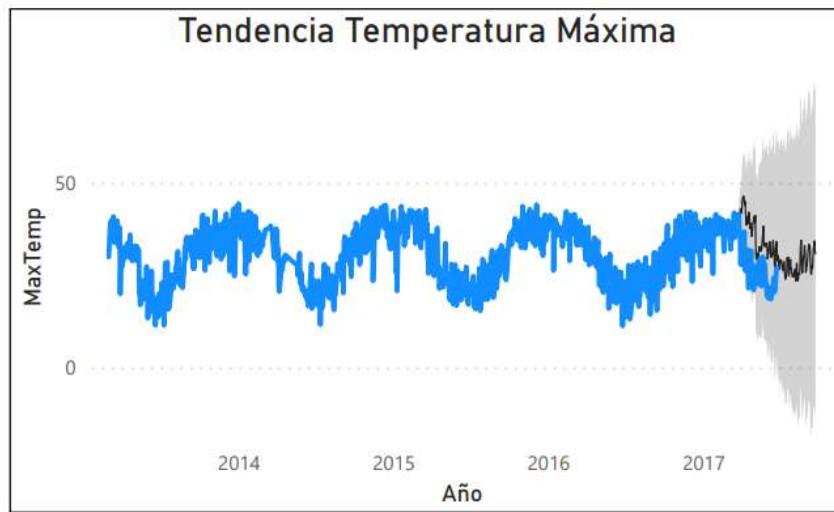
Ranking Dirección del viento

El gráfico de barras nos muestra el ranking de la dirección del viento, filtrando con los 6 primeros valores. Esta información es de suma importancia, ya que podemos determinar y recomendar en qué dirección colocar el viñedo para aprovechar los vientos o por el contrario para evitarlos. La dirección del viento que se presenta con mayor frecuencia es al Este.



Gráfico de Tendencia Temperatura Máxima

Visualizamos la tendencia que presenta la temperatura máxima a través del tiempo. También en la parte derecha del gráfico, podemos generar una predicción de como continuará la tendencia (línea negra), la parte sombreada con color gris es la zona de confianza. Se puede ver que la tendencia continúa como viene suscitándose.



Conclusión

En la fase de Entendimiento del negocio vimos los objetivos que se tienen, un resumen del problema y lo importante que es conocer al respecto. La fase de comprensión de datos de CRISP-DM implica estudiar más de cerca los datos disponibles de minería. Este paso es esencial para evitar problemas inesperados durante la siguiente fase (preparación de datos) que suele ser la fase más larga de un proyecto.

La comprensión de datos implica acceder a los datos y explorarlos con la ayuda de tablas y gráficos. La preparación de datos es uno de los aspectos más importantes y con frecuencia que más tiempo exigen en la minería de datos. Esta primera evaluación nos permitió conocer los primeros pasos en Crisp-dm, si bien es cierto que nos encontramos con algunos inconvenientes, los pudimos solucionar y tomamos en consideración algunas de las observaciones dadas por el docente.

La etapa 4 consta de la creación de modelos. En nuestro caso en particular, realizamos dos tipos de modelos: Clasificación y regresión. Nos encontramos con dificultades de hardware, pero lo pudimos solucionar.

Como penúltima etapa, la evaluación. Revisamos y comparamos las métricas dadas por los distintos modelos; Con esto en mente, seleccionamos los dos modelos a utilizar según el costo vs beneficio y tiempo de implementación.

La última etapa, el despliegue, consideramos el aprendizaje aprendido y los gráficos que serán presentados a los interesados con los que mostramos recomendaciones de acción. Es una etapa en la que se muestra el consolidado del proyecto.

Como supuesto, consideramos que nos contactaba una empresa que se encarga de hacer vinos con su propio viñedo (VVB) y querían instalarse en Australia. Para esto nos solicitaban la predicción de temperaturas máximas y predicción de lluvias. Para llevar a cabo, la solicitud en su totalidad; investigamos nuevas tecnologías que nos podrían servir en próximos proyectos similares. Fue importante considerar otras variables para la creación de los vinos como la humedad o la altitud de la ciudad.