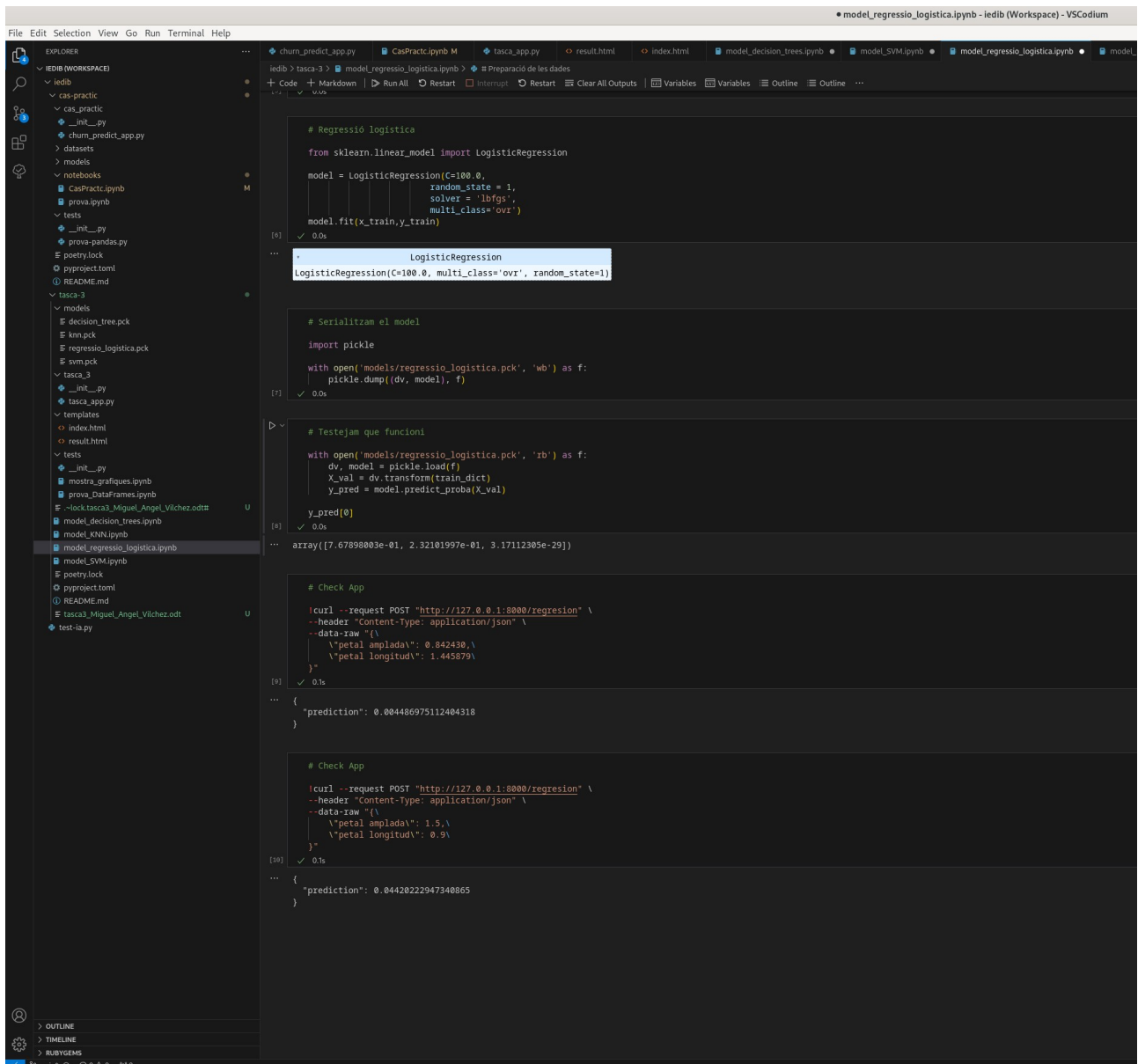


## URL del GitHub

<https://github.com/mangelvil/tasca-3>

## Estructura del projecte al VSCodium( versió lliure de telemetry/tracking per part de Microsoft ).



The screenshot shows the VS Code interface with a project named 'tasca-3' open. The Explorer panel on the left displays the project structure, including files like 'churn\_predict\_app.py', 'model\_predict\_app.py', 'models', 'notebooks', 'tests', and 'tasca-3'. The main editor area shows the code for 'model\_predict\_app.py', which includes a logistic regression model using sklearn, saving the model to a pickle file, and testing it with a sample input.

```
# Regressió logística

from sklearn.linear_model import LogisticRegression

model = LogisticRegression(C=100.0,
                           random_state = 1,
                           solver = 'lbfgs',
                           multi_class='ovr')
model.fit(x_train,y_train)

[6] ✓ 0.0s

...

LogisticRegression
LogisticRegression(C=100.0, multi_class='ovr', random_state=1)

# Serialitza el model

import pickle

with open('models/regressio_logistica.pkl', 'wb') as f:
    pickle.dump(dv, model), f)

[7] ✓ 0.0s

# Testejem que funcioni

with open('models/regressio_logistica.pkl', 'rb') as f:
    dv, model = pickle.load(f)
    X_val = dv.transform(train_dict)
    y_pred = model.predict_proba(X_val)

y_pred[0]

[8] ✓ 0.0s

...

array([[7.67898803e-01, 2.32101997e-01, 3.17112305e-29]])

# Check App

!curl --request POST "http://127.0.0.1:8080/regression" \
--header "Content-Type: application/json" \
--data-raw "{
  \"petal amplada\": 0.842430,
  \"petal longitud\": 1.445879
}"

[9] ✓ 0.1s

...

{
  "prediction": 0.004486975112404318
}

# Check App

!curl --request POST "http://127.0.0.1:8080/regression" \
--header "Content-Type: application/json" \
--data-raw "{
  \"petal amplada\": 1.5,
  \"petal longitud\": 0.9
}"

[10] ✓ 0.1s

...

{
  "prediction": 0.04420222947340865
}
```

## Contingut de l'arxiu pyproject.toml

```
...  churn_predict_app.py  CasPractc.ipynb M  tasca_app.py  result.html

iedib > tasca-3 > pyproject.toml
1  [tool.poetry]
2  name = "tasca-3"
3  version = "0.1.0"
4  description = ""
5  authors = ["Miguel Ángel Vílchez López <mangel@matronica.eu>"]
6  readme = "README.md"
7
8  [tool.poetry.dependencies]
9  python = "^3.11"
10 numpy = "^1.26.3"
11 matplotlib = "^3.8.2"
12 pandas = "^2.1.4"
13 scikit-learn = "^1.3.2"
14 flask = "^3.0.0"
15 ipykernel = "^6.28.0"
16
17
18 [build-system]
19 requires = ["poetry-core"]
20 build-backend = "poetry.core.masonry.api"
21
```

**Execució del client que fa les peticions als models, on es vegi també la resposta obtinguda. Comenta també el resultat. Com a mínim has d'incloure les captures corresponents a dues peticions per a cada un dels 4 models: totes han d'aparèixer en el document, amb la seva resposta i el seu comentari del resultat.**

Aixó de poder guardar un entrenament per poder usarlo després es francament increïble.  
He usat el mateix Jupyter per fer les peticions per CURL.  
Tambè he fet una interfície web per poder utilitzar els entrenaments dels diferents models.  
Adjunt captures de l'interfície web.

He passat els mateixos valors de longitud i d'amplada a tots els models.

Els primers valors son 1.4, 0.2	→ Correspon a un dels valors del iris 0 (Iris Setosa).
Els segons valors son 3.8, 1.0	→ Correspon a un dels valors del iris 1 (Iris Versicolour).
Els tercers valors son 5.7, 2.3	→ Correspon a un dels valors del iris 2 (Iris Virginica).
Els quarts valors son 2.1, 0.9	→ He encontrat un valor que dona diferent depenent del model.

He possat els mateix valors que dona el dataset de l'iris el primer ha de donar 0, el segon 1 i el tercer 2, però no hi ha manera aconseguir que doni 2 de resultat, hi he provat molts de valors alts i baixos.

Respecte als models.

El de regresió logística dona una predicció que imagin que hauria de fer un "round" per a que doni la resposta esperada. Ho deix així porque es vegi diferent als altres.  
Respecte al quart valor s'aproxima a 0.

El SVM dona unes prediccions parescudes al de regresió logística, més o manco igual, si arrodonim ens donarà els mateixos valors que la resta.  
Respecte al quart valor s'aproxima a 0.

El de decision tree dona una predicció esperada excepte l'últim que hauria de donar 2.  
Respecte al quart valor dona 1.

El de KKN dona el mateix que l'anterior, predicció esperada en el primer i segon, però el tercer hauria de donar 2 i no ho fa.  
Respecte al quart valor dona 0.

El model Decision Tree dona un resultat diferent a la resta en el quart valor, aixó és porque just he possat una combinació que en aquest model perteneix a l'1 i a la resta perteneix al 0. Podem observar les gràfiques i vore que agafen àrees diferents.

```

model.fit(x_train,y_train)
✓ 0.0s

* LogisticRegression
LogisticRegression(C=100.0, multi_class='ovr', random_state=1)

# Serialitzam el model

import pickle

with open('models/regressio_logistica.pck', 'wb') as f:
    pickle.dump((dv, model), f)
✓ 0.0s

# Check App

!curl --request POST "http://127.0.0.1:8000/regresion" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 1.4,\
  \"petal longitud\": 0.2\
}"
✓ 0.1s

{
  "prediction": 0.18102254676173993
}

# Check App

!curl --request POST "http://127.0.0.1:8000/regresion" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 3.8,\
  \"petal longitud\": 1.1\
}"
✓ 0.1s

{
  "prediction": 0.9979685429513531
}

# Check App

!curl --request POST "http://127.0.0.1:8000/regresion" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 5.7,\
  \"petal longitud\": 2.3\
}"
✓ 0.1s

{
  "prediction": 0.1894619915691005
}

# Check App

!curl --request POST "http://127.0.0.1:8000/regresion" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 2.1,\
  \"petal longitud\": 0.9\
}"
✓ 0.1s

{
  "prediction": 0.09458462743643795
}

```

```

model = SVC(kernel='linear', C=1.0, random_state=1, probability=True)
model.fit(x_train, y_train)
✓ 0.0s

* SVC
SVC(kernel='linear', probability=True, random_state=1)

# Serialitzam el model

import pickle

with open('models/svm.pck', 'wb') as f:
    pickle.dump((dv, model), f)
✓ 0.0s

# Check App

!curl --request POST "http://127.0.0.1:8000/svm" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 1.4,\
  \"petal longitud\": 0.2\
}"
✓ 0.1s

{
  "prediction": 0.023440141400982367
}

# Check App

!curl --request POST "http://127.0.0.1:8000/svm" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 3.8,\
  \"petal longitud\": 1.1\
}"
✓ 0.1s

{
  "prediction": 0.9472113053524546
}

# Check App

!curl --request POST "http://127.0.0.1:8000/svm" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 5.7,\
  \"petal longitud\": 2.3\
}"
✓ 0.1s

{
  "prediction": 0.007849189499302271
}

# Check App

!curl --request POST "http://127.0.0.1:8000/svm" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 2.1,\
  \"petal longitud\": 0.9\
}"
✓ 0.1s

{
  "prediction": 0.1959817595577995
}

```

```

model = DecisionTreeClassifier(criterion='gini', max_depth=4,
                               random_state=1)
model.fit(x_train, y_train)
✓ 0.0s

* DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=1)

# Serialitzam el model
import pickle

with open('models/decision_tree.pck', 'wb') as f:
    pickle.dump((dv, model), f)
✓ 0.0s

# Check App
!curl --request POST "http://127.0.0.1:8000/decision" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 1.4,\
  \"petal longitud\": 0.2\
}"
✓ 0.1s

{
  "prediction": 0.0
}

# Check App
!curl --request POST "http://127.0.0.1:8000/decision" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 3.8,\
  \"petal longitud\": 1.1\
}"
✓ 0.1s

{
  "prediction": 1.0
}

# Check App
!curl --request POST "http://127.0.0.1:8000/decision" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 5.7,\
  \"petal longitud\": 2.3\
}"
✓ 0.1s

{
  "prediction": 0.0
}

# Check App
!curl --request POST "http://127.0.0.1:8000/decision" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 2.1,\
  \"petal longitud\": 0.9\
}"
✓ 0.1s

{
  "prediction": 1.0
}

```

```

* KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)

# Serialitzam el model
import pickle

with open('models/knn.pck', 'wb') as f:
    pickle.dump((dv, model), f)
✓ 0.0s

# Check App
!curl --request POST "http://127.0.0.1:8000/knn" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 1.4,\
  \"petal longitud\": 0.2\
}"
✓ 0.1s

{
  "prediction": 0.0
}

# Check App
!curl --request POST "http://127.0.0.1:8000/knn" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 3.8,\
  \"petal longitud\": 1.1\
}"
✓ 0.1s

{
  "prediction": 1.0
}

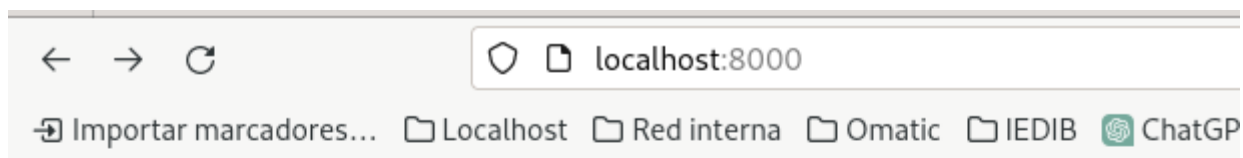
# Check App
!curl --request POST "http://127.0.0.1:8000/knn" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 5.7,\
  \"petal longitud\": 2.3\
}"
✓ 0.1s

{
  "prediction": 0.0
}

# Check App
!curl --request POST "http://127.0.0.1:8000/knn" \
--header "Content-Type: application/json" \
--data-raw "{\
  \"petal amplada\": 2.1,\
  \"petal longitud\": 0.9\
}"
✓ 0.1s

{
  "prediction": 0.0
}

```

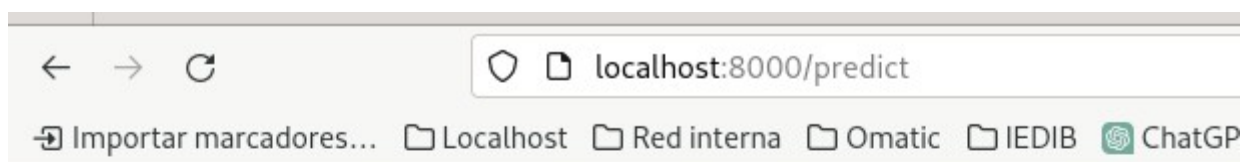


## Introduïu els valors que vols:

Amplada petal:

Longitud petal:

Model:



## Resultat:

Amplada Petal: 2.1

Longitud Petal: 0.9

Model: knn

Predicció: 0.0

[Go back](#)