



# Jaqqot5 tutorials

Jaqqot 5: How to deploy a predictive model using the jaqqotpy library

USE:	How to deploy a predictive model using the jaqqotpy library
VERSION:	V.1.0
CONTACT DETAILS:	Haralambos Sarimveis: <a href="mailto:hsarimv@central.ntua.gr">hsarimv@central.ntua.gr</a> Pantelis Karatzas: <a href="mailto:pantelispanka@gmail.com">pantelispanka@gmail.com</a> Philip Doganis: <a href="mailto:filipposd@gmail.com">filipposd@gmail.com</a>

# INTRODUCTION

Jaqpote 5 is a user-friendly web-based e-infrastructure that allows model developers to deploy their predictive models and share them through the web. The Jaqpote 5 GUI directs the model developers to further document their models in a way that can be easily understood and used by end-users with little or no experience on machine learning and statistical analysis. The GUI also allows the end-users to apply the models on their own data for validation and/or prediction purposes and the results are collected and visualised in automatically generated tables, graphs and reports. All major machine learning and statistical data-driven algorithms are supported in Jaqpote 5, by integrating popular libraries such as the Python Scikit-learn and the R Caret libraries. Jaqpote 5 has been designed as a generic modelling and machine learning web platform, but particular emphasis is given on serving the needs of the chemo/bio/nano/pharma/ communities by integrating QSAR, biokinetics, dose-response and read-across models. Jaqpote 5 has been developed by the [Unit of Process Control and Informatics](#) in the School of Chemical Engineering at the National Technical University of Athens.

This document provides a tutorial on how to deploy a model in Jaqpote 5 using the jaqpote library. The resource has been made available at <https://app.jaqpote.org/>.

# DEPLOYING A PREDICTIVE MODEL USING THE JAQPOTPY LIBRARY

Jaqpot 5 is currently integrated with the entire Scikit-learn python library (<https://scikit-learn.org/stable/>) which is the most comprehensive and perhaps the most popular open source library for machine learning, data mining and data analysis. There are plans to integrate algorithms from the R language caret library (<https://cran.r-project.org/web/packages/caret/caret.pdf>) as well as techniques from Julia machine learning libraries, like JuliaML (<https://github.com/JuliaML>).

The main tool developed by NTUA for integrating the Scikit-learn set of algorithms is the jaqpotpy library, which lets the user create a machine learning model in the python environment of his choice and the deployment the model over the web.

To follow this tutorial the user should:

- 1) Have a Jaqpot 5 account. For more information, please read the tutorial on how to login to Jaqpot 5
- 2) The Jaqpotpy library should be installed as a pypi package:  

```
pip install jaqpotpy
```
- 3) One csv file containing the data for a particular modelling example should be available. The file is provided as supplementary material in this deliverable and its name is:  
`70_reduced.csv`

After the end of the tutorial, the user should arrive to a web service similar to the one hosted in the following URL: <https://app.jaqpot.org/model/uxRBCMsV9lkSQT1Kw7km>. Thus model is available through the NanoCommons organisation. For more information about Jaqpot 5 organisations please read the specific tutorial on using and managing organisations in Jaqpot 5.

In the Appendix at the end of this tutorial, the reader can find detailed information about all the functionalities of the Jaqpot 5 library.

The modeller can use the algorithm of her/his choice to fit the best possible model to the available training data and validate the model based on validation statistics. She/he can also use optimal machine learning tools provided in Python, such as TPOT, a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming (<https://github.com/EpistasisLab/tpot>).

In this tutorial we will demonstrate how to reproduce and publish in Jaqpot5, a Linear nanoQSAR model predicting Solubility of C60 Fullerene in Various Solvents, with just a few lines of code in a Jupyter notebook. The model has been originally presented in the following publication: Farhad Gharagheizi & Reza Fareghi Alamdari (2008) A Molecular-Based Model for Prediction of Solubility of C60 Fullerene in Various Solvents, Fullerenes, Nanotubes, and Carbon Nonstructures, 16:1, 40-57, DOI: 10.1080/15363830701779315.

Import the jaqpotpy library and various components from the pandas and scikit-learn python packages:

```
import pandas as pd

from jaqpotpy import Jaqpot

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, GridSearchCV, RandomizedSearchCV

df=pd.read_csv('70_model_reduced.csv') # Reads the data

print(list(df)) # Prints the headers of all columns
```

Read the data and print the headers of all columns:

```
df=pd.read_csv('70_model_reduced.csv') # Reads the data  
print(list(df)) # Prints the headers of all columns
```

The response is a list with the headers of all columns:

```
['Solvents', 'piPC03', 'ATS1m', 'Seigp', 'More23e', 'H1m', 'logS Exp.']
```

Define the independent variables and the end-point to be predicted and split randomly the dataset into training and test sets consisting of 75% and 25% of the data respectively:

```
Xall=df[['piPC03', 'ATS1m', 'Seigp', 'More23e', 'H1m']] # Defines the columns that will be used as independent features  
Yall=df['logS Exp.'] # Defines the end-point  
X_train, X_test, Y_train, Y_test = train_test_split(Xall, Yall, train_size=0.75, test_size=0.25, random_state=1)  
# Splits the data into training and test sets
```

Develop a pipeline consisting of scaling the data first and then apply the multiple linear regression algorithm:

```
stepslinear = [('scaler', MinMaxScaler()), ('MLR', LinearRegression())]  
pipelinelinear = Pipeline(stepslinear) # define the pipeline object.
```



Train the model on the training set, compute and print the  $R^2$  statistic on the training set, the test set and on the full set. Finally perform 5-fold cross validation test on the training observation.

```
pipelinelinear.fit(X_train, Y_train)
print('Training score: ', pipelinelinear.score(X_train, Y_train))
print('Testing score: ', pipelinelinear.score(X_test, Y_test))
print('Total score: ', pipelinelinear.score(Xall, Yall)) #Trains the model and prints R^2 statistics

cross_val_score(estimator=pipelinelinear, X=X_train, y=Y_train, cv=5, n_jobs=-1) #Performs a 5-fold cross
validation
```

The response is a list with the calculated statistics:

Training score: 0.8994088488271355

Testing score: 0.9043040438111096

Total score: 0.9034772311898356

array([0.91906039, 0.88995619, 0.90445436, 0.86506266, 0.62316459])

The results are very satisfactory and very similar to the one reported in the paper, where the model has been presented. The following commands are used to deploy the model as a web service in Jaqpot 5. First the user is prompted to enter his username and password:

```
jaqpot = Jaqpot("https://api.jaqpot.org/jaqpot/services/")  
jaqpot.request_key_safe()
```

Only a single command is needed to deploy the model into Jaqpot 5:

```
jaqpot.deploy_pipeline(pipelinelinear,Xall,Yall,"Linear Model for Predicting Solubility of C60 Fullerenes in  
Various Solvents","Linear Model","linearmodel")
```

The response is the unique Jaqpot 5 URL on which the model is hosted.

Optionally, the user can create a Predictive Model Markup Language (PMML) representation of the model to be included as additional information in the web service:

```
from sklearn2pmml.pipeline import PMMLPipeline

from sklearn2pmml.pipeline import PMMLPipeline
pipelinepmmllinear = PMMLPipeline([
    ("scaler", MinMaxScaler()), ("MLR", LinearRegression())
])
pipelinepmmllinear.fit(X_train, Y_train)

from sklearn2pmml import sklearn2pmml

sklearn2pmml(pipelinepmmllinear, "SolubilityC60linear.pmml", with_repr = True)
```

The model developer can now enter the Jaqpot UI to add any other information about the model (for example detailed description, standard reports like Quantitative Model Reporting Format (QMRF), PMML representations, ontological annotations etc.)

The overview tab: The overview tab opens a markdown free-text editor where the user can include any information about the model. The editing mode can be activated by clicking the red icon on the bottom right of the page (Figure 1) and is deactivated by clicking on the floppy disk icon (Figure 2). The screenshot in Figure 1 is from our full implementation of the model, where we are providing links to the full dataset, the training dataset and the test data sets, a link to a full QMRF report generated using the QMRF editor provided by JRC: <https://sourceforge.net/projects/qmrf/> the report an editable version of the QMRF report and the PMML representation of the model. For uploading dataset to Jaqpot 5 please read the tutorial on uploading datasets.



The screenshot shows the Jaqpot interface with the 'Overview' tab selected. The model title is 'Linear nanoQSAR model predicting Solubility of C60 Fullerene in Various Solvents'. The description includes the citation: 'The model is provided in the following publication: Farhad Gharagheizi & Reza Fareghi Alamdari (2008) A Molecular-Based Model for Prediction of Solubility of C60 Fullerene in Various Solvents, Fullerenes, Nanotubes, and Carbon Nonstructures, 16:1, 40-57, DOI: 10.1080/15363830701779315'. Below the description are four links: 'Full dataset is available in this link', 'Training dataset is available in this link', 'Test dataset is available in this link', and 'A downloadable QMRF Report is available in this link'. The 'QMRF Report' section is partially visible, showing '1. QSAR Identifier' and '1.1. QSAR identifier (title):'. A red pencil icon in the bottom right corner is highlighted with a blue arrow, indicating the edit mode activation button.

**Figure 1.** Overview tab (activating the editing mode).

Jaqpot

Overview Features Predict / Validate Discussion Archive

**MODEL**  
Title: Linear Model for Predicting Solubility of C60 Fullerenes in Various Solvents  
Owner: hsarimv  
Description  
Linear Model for Predicting Solubility of C60 Fullerenes in Various Solvents

Edit overview \*

```
# Linear nanoQSAR model predicting Solubility of C60 Fullerene in Various Solvents.

The model is provided in the following publication: Farhad Gharagheizi & Reza Fareghi Alamdari (2008) A Molecular-
Based Model for Prediction of Solubility of C60 Fullerene in Various Solvents, Fullerenes, Nanotubes, and Carbon
Nonstructures, 16.1, 40-57, DOI: 10.1080/15363830701779315

[Full dataset is available in this link](https://app.jaqpot.org/dataset/cjCXIjkX0kBbjelkHMZuNg)

[Training dataset is available in this link](https://app.jaqpot.org/dataset/MYs9MrJuJmGSjGhwq3DE72)

[Test dataset is available in this link](https://app.jaqpot.org/dataset/RHokDX86xSN3BLj9yL2E5x)

[A downloadable QMRF Report is available in this link](https://github.com/ntua-unit-of-control-and-informatics/QSAR-
Models/blob/master/Lineal%20Model%20Predicting%20Solubility%20of%20C60%20fullerenes.xml.pdf)

***
# QMRF Report

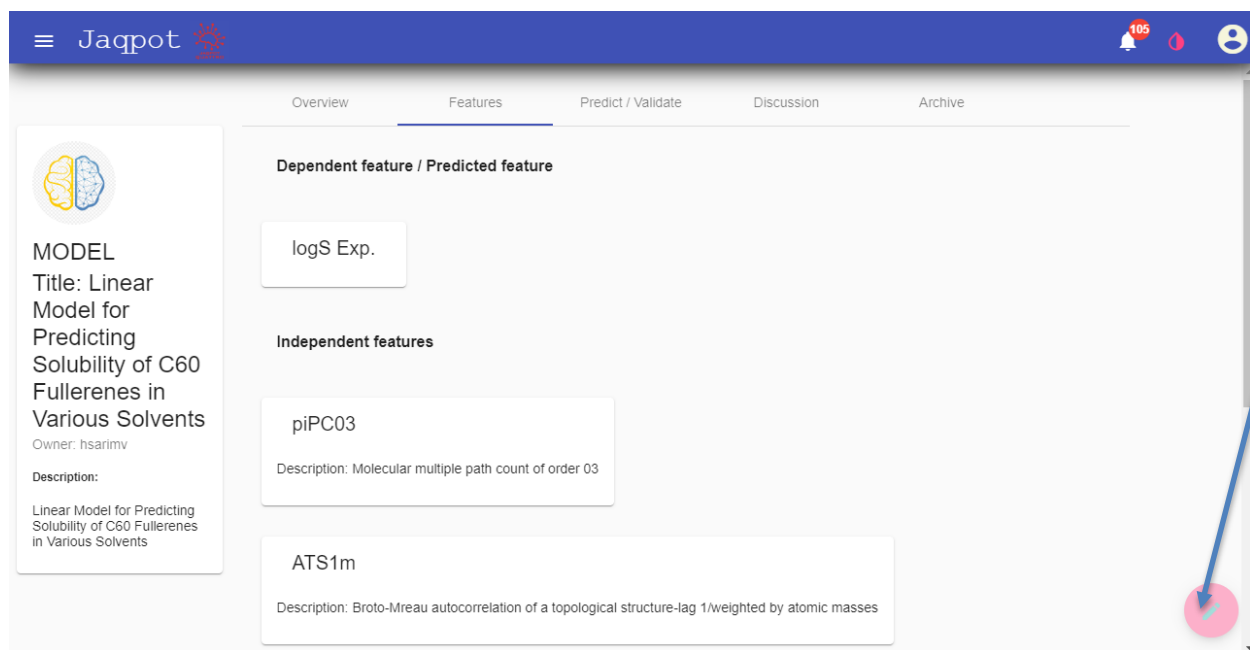
## 1. QSAR Identifier

### 1.1. QSAR identifier (title):

Linear nanoQSAR model predicting Solubility of C60 Fullerene in Various
Solvents
```

**Figure 2.** Overview tab (deactivating the editing mode).

The features tab: Here the model creator can provide specific information about the independent features and the end-point of the model: descriptions, units and ontological classes, which will allow the model to understand data sets that are ontologically annotated automatically. Below (Figure 3) we provide a screenshot of the data tab under the C60 solubility model. Like in the overview tab, the user can enter or change all that information by clicking the red icon on the bottom right of the page, which activates the editing mode (Figure 4)



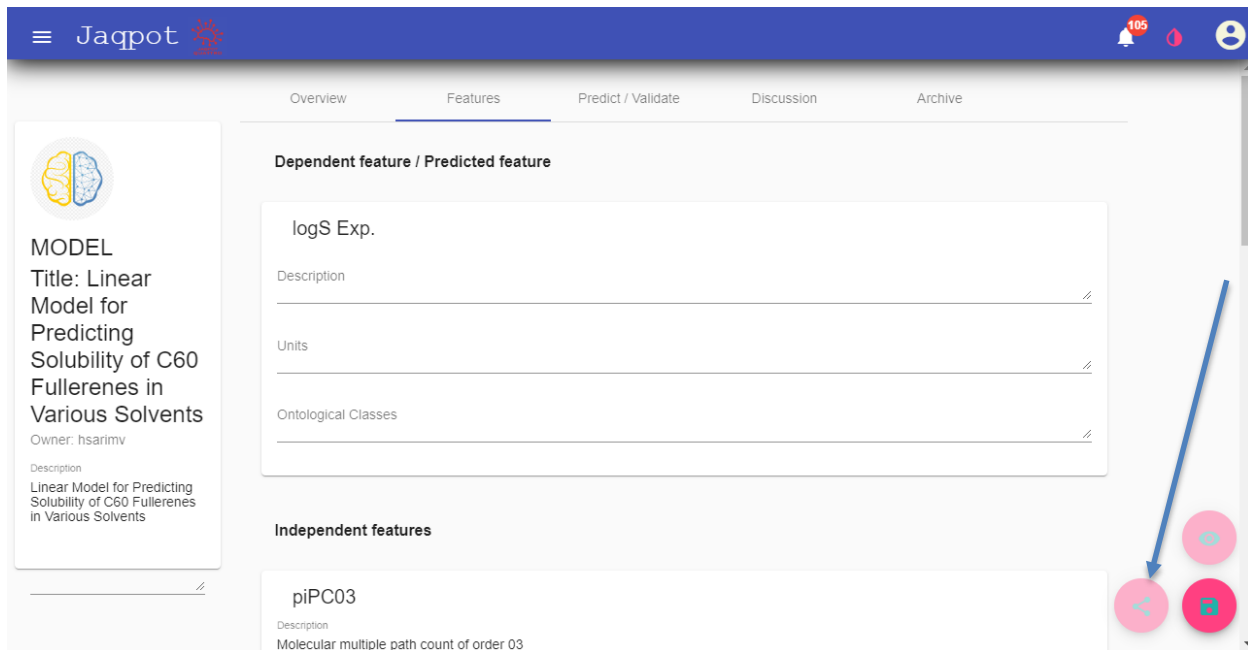
The screenshot displays the Jaqpot web application interface. At the top, there is a blue header with the Jaqpot logo and navigation icons. Below the header, a horizontal menu contains tabs for 'Overview', 'Features', 'Predict / Validate', 'Discussion', and 'Archive'. The 'Features' tab is currently selected. On the left side, a sidebar displays model information: 'MODEL', 'Title: Linear Model for Predicting Solubility of C60 Fullerenes in Various Solvents', 'Owner: hsarimv', and 'Description: Linear Model for Predicting Solubility of C60 Fullerenes in Various Solvents'. The main content area is divided into two sections: 'Dependent feature / Predicted feature' containing a box with 'logS Exp.', and 'Independent features' containing two boxes. The first independent feature is 'piPC03' with the description 'Molecular multiple path count of order 03'. The second is 'ATS1m' with the description 'Broto-Mreau autocorrelation of a topological structure-lag 1/weighted by atomic masses'. A red circular icon with a white pencil is located in the bottom right corner of the main content area, and a blue arrow points to it from the right edge of the image.

**Figure 3.** The features tab (entering information and ontological annotations for the independent variables and the end-point predicted by the model).

The screenshot shows the Jaqpot web application interface. At the top, there is a blue header with the Jaqpot logo and navigation icons. Below the header, a navigation bar contains tabs for Overview, Features, Predict / Validate, Discussion, and Archive. The 'Features' tab is active. On the left side, there is a sidebar for the model, displaying a brain icon, the title 'MODEL', and the full title 'Linear Model for Predicting Solubility of C60 Fullerenes in Various Solvents'. The main content area is divided into two sections: 'Dependent feature / Predicted feature' and 'Independent features'. The 'Dependent feature' section contains a form for 'logS Exp.' with fields for Description, Units, and Ontological Classes. The 'Independent features' section contains a form for 'piPC03' with a description 'Molecular multiple path count of order 03'. On the right side of the main content area, there are three circular icons: a pink one with a gear, a pink one with a share symbol, and a red one with a document symbol.

**Figure 4.** The features tab in editing mode.

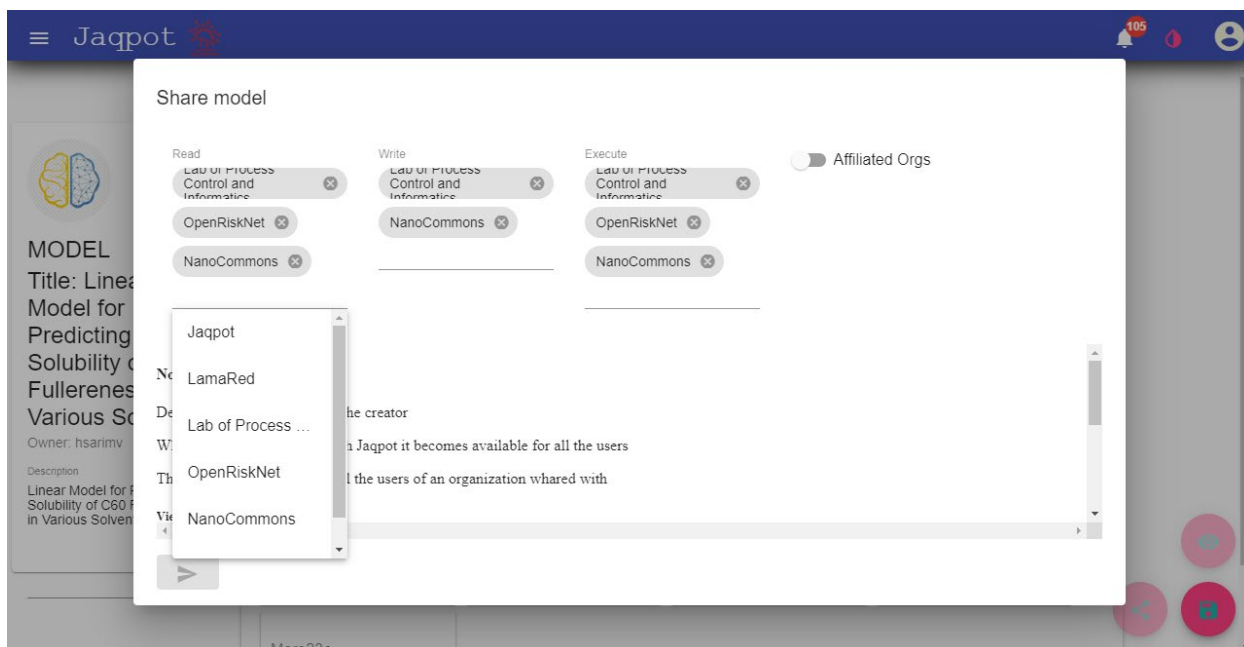
The model is now complete. The user can share the model with his partners or the community through the Jaqpot organisations. To do that the user clicks on the share button, which is available in all tabs.



**Figure 4.** The features tab in editing mode.



He can choose to share the models in various levels (read, write, execute) with organisations where he is a member. For more information about Jaqpot 5 organisations please read the specific tutorial on using and managing organisations in Jaqpot 5.



**Figure 5.** Sharing the model with organisations.

## Appendix. Guidelines on installing and using the *jaqpotpy* library

### Installation

In order to use *jaqpotpy*, users need to install it first. Installation can be executed conveniently as a `pip` package.

```
pip install jaqpotpy
```

### Usage and initialization

#### Import Jaqpot

After installation, Jaqpot needs to be imported with the following command:

```
from jaqpotpy import Jaqpot
```

#### Initialize Jaqpotpy on the services where jaqpot lives.

```
jaqpot = Jaqpot("https://api.jaqpot.org/jaqpot/services/")
```

#### User authentication by Jaqpot

In order to access *jaqpot* services, first the authentication of the user is required. First it is necessary to define the web location of the Jaqpot instance being used.

```
jaqpot = Jaqpot("https://api.jaqpot.org/jaqpot/services/")
```

The following command will send your username and password, execute your login and set the api key that is needed:

```
jaqpot.request_key('username', 'password')
```

Same as above, this command hides the password if *jaqpot* is used through a jupyter notebook etc. and initiates a prompt for your username and password, only visible by the user:

```
jaqpot.request_key_safe()
```

Alternatively, for users that have logged in through google or github it is possible to login with the use of an API key. At the account page, the user can find an api key that can be used in order to have access to the services and send it to Jaqpot by substituting the `api_key` field. Please note that these keys have short life and should be updated on each login.

```
jaqpot.set_api_key("api_key")
```

## Deploy your models!

You can use the commands below, customised per model type, in order to make models trained with scikit-learn algorithms available as web services through Jaqpot. Please note:

- all models should be trained with variables that are pandas dataframes
- when calling a `jaqpot.deploy` function you should use exactly the same variables used to train the model
- The Y variable (prediction endpoint) should have an index.

### 1. `deploy_linear_model()`

```
jaqpot.deploy_linear_model()
```

Lets you deploy linear models are created with scikit-learn. Bellow there is a list for the produced models that can be deployed with this function:

- `linear_model.ARDRRegression()`
- `linear_model.BayesianRidge()`
- `linear_model.ElasticNet()`
- `linear_model.ElasticNetCV()`
- `linear_model.HuberRegressor()`
- `linear_model.Lars()`
- `linear_model.LarsCV()`
- `linear_model.Lasso()`
- `linear_model.LassoCV()`
- `linear_model.LassoLars()`
- `linear_model.LassoLarsCV()`
- `linear_model.LassoLarsIC()`
- `linear_model.LinearRegression()`
- `linear_model.LogisticRegression()`
- `linear_model.LogisticRegressionCV()`
- `linear_model.MultiTaskLasso()`
- `linear_model.MultiTaskElasticNet()`
- `linear_model.MultiTaskLassoCV()`
- `linear_model.MultiTaskElasticNetCV()`
- `linear_model.OrthogonalMatchingPursuit()`
- `linear_model.OrthogonalMatchingPursuitCV()`
- `linear_model.PassiveAggressiveClassifier()`
- `linear_model.PassiveAggressiveRegressor()`
- `linear_model.Perceptron()`

- `linear_model.RANSACRegressor()`
- `linear_model.Ridge()`
- `linear_model.RidgeClassifier()`
- `linear_model.RidgeClassifierCV()`
- `linear_model.RidgeCV()`
- `linear_model.SGDClassifier()`
- `linear_model.SGDRegressor()`
- `linear_model.TheilSenRegressor()`
- `linear_model.enet_path()`
- `linear_model.lars_path()`
- `linear_model.lasso_path()`
- `linear_model.logistic_regression_path()`
- `linear_model.orthogonal_mp()`
- `linear_model.orthogonal_mp_gram()`
- `linear_model.ridge_regression()`

`deploy_linear_model()` parameters are:

- **model** :{is a sklearn trained model} A trained model that occurs from the `sklearn.linear_model` family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).
- **y** : {is a pandas dataframe} The dataframe that is used to train the model (y variables).
- **title** : {is a String} The title of the model
- **description** : {is a String} The description of the model
- **algorithm** : {is a String} The algorithm that the model implements string

The id of the model is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

### Example usage

```
from jaqpotpy import Jaqpot
import pandas as pd
from sklearn import linear_model

df = pd.read_csv('/path/train.csv')
X = df[['Pclass', 'SibSp', 'Parch', 'Fare']]
y = df['Survived']

clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').fit(X, y)

jaqpot.deploy_linear_model(clf, X, y, title="Sklearn 2", description="Logistic regression model from python for the titanic dataset",
                           algorithm="logistic regression")
```

On the above example a linear model (in our case a logistic regression) is created and deployed on Jaqpot. The dataset is read as a pandas dataframe (a requirement for Jaqpot 5) and the X and y dataframes are created, on which the algorithm is trained and the model is created.

## 2. `deploy_cluster()`

Allows deployment of cluster models that are created from scikit-learn algorithms:

- `cluster.AffinityPropagation()`
- `cluster.AgglomerativeClustering()`
- `cluster.Birch()`
- `cluster.DBSCAN()`
- `cluster.FeatureAgglomeration()`
- `cluster.KMeans()`
- `cluster.MinibatchKMeans()`
- `cluster.MeanShift()`
- `cluster.SpectralClustering()`

`jaqpot.deploy_deploy_cluster()` parameters are:

- **model** : {is a sklearn trained model} a trained model that occurs from the `sklearn.linear_model` family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).
- **title**: {is a String} The title of the model
- **description**: {is a String} The description of the model
- **algorithm**: {is a String} The algorithm that the model implements string

The id of the model is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

## 3. `deploy_ensemble()`

Allows deployment of cluster models that are created from scikit-learn algorithms:

- `ensemble.AdaBoostClassifier()`
- `ensemble.AdaBoostRegressor()`
- `ensemble.BaggingClassifier()`
- `ensemble.BaggingRegressor()`
- `ensemble.ExtraTreesClassifier()`
- `ensemble.ExtraTreesRegressor()`
- `ensemble.GradientBoostingClassifier()`
- `ensemble.GradientBoostingRegressor()`
- `ensemble.IsolationForest()`
- `ensemble.RandomForestClassifier()`
- `ensemble.RandomForestRegressor()`
- `ensemble.RandomTreesEmbedding()`
- `ensemble.VotingClassifier()`

`jaqpot.deploy_ensemble()` parameters are:

- **model** : {is a sklearn trained model} is a trained model that occurs from the `sklearn.linear_model` family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).

- **y** : {is a pandas dataframe} The dataframe that is used to train the model (y variables).
- **title**: {is a String} The title of the model
- **description**: {is a String} The description of the model
- **algorithm**: {is a String} The algorithm that the model implements string

The id of the model is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

#### 4. **deploy\_naive\_bayess()**

Allows deployment of naive\_bayes models that are created from scikit-learn algorithms:

- naive\_bayes.BernoulliNB()
- naive\_bayes.GaussianNB()
- naive\_bayes.MultinomialNB()
- naive\_bayes.ComplementNB()

jaqpot.deploy\_naive\_bayess() parameters are:

- **model** : {is a sklearn trained model} is a trained model that occurs from the sklearn.linear\_model family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).
- **y** : {is a pandas dataframe} The dataframe that is used to train the model (y variables).
- **title**: {is a String} The title of the model
- **description**: {is a String} The description of the model
- **algorithm**: {is a String} The algorithm that the model implements string

The id of the model is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

#### 5. **deploy\_nearest\_neighbors()**

Allows deployment of nearest\_neighbors models that are created from scikit-learn algorithms:

- neighbors.KNeighborsClassifier()
- neighbors.KNeighborsRegressor()
- neighbors.LocalOutlierFactor()
- neighbors.RadiusNeighborsClassifier()
- neighbors.RadiusNeighborsRegressor()
- neighbors.NearestCentroid()
- neighbors.NearestNeighbors()
- neighbors.kneighbors\_graph()
- neighbors.radius\_neighbors\_graph()

jaqpot.deploy\_nearest\_neighbors() parameters are:

- **model** : {is a sklearn trained model} is a trained model that occurs from the sklearn.linear\_model family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).

- **y** : {is a pandas dataframe} The dataframe that is used to train the model (y variables).
- **title**: {is a String} The title of the model
- **description**: {is a String} The description of the model
- **algorithm**: {is a String} The algorithm that the model implements string

If y is empty, Jaqpot generates an empty dataframe with the title of the predicted feature.

The id of the model is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

## 6. `deploy_neural_network()`

Allows deployment of `neural_network` models that are created from scikit-learn algorithms:

- `neural_network.BernoulliRBM()`
- `neural_network.MLPClassifier()`
- `neural_network.MLPRegressor()`

`jaqpot.deploy_neural_network()` parameters are:

- **model** : {is a sklearn trained model} is a trained model that occurs from the `sklearn.linear_model` family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).
- **y** : {is a pandas dataframe} The dataframe that is used to train the model (y variables).
- **title**: {is a String} The title of the model
- **description**: {is a String} The description of the model
- **algorithm**: {is a String} The algorithm that the model implements string

The id of the model is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

## 7. `deploy_svm()`

Allows deployment of `svm` models that are created from scikit-learn algorithms:

- `svm.LinearSVC()`
- `svm.LinearSVR()`
- `svm.NuSVC()`
- `svm.NuSVR()`
- `svm.OneClassSVM()`
- `svm.SVC()`
- `svm.SVR()`
- `svm.l1_min_c()`

`jaqpot.deploy_svm()` parameters are:

- **model** : {is a sklearn trained model} is a trained model that occurs from the `sklearn.linear_model` family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).
- **y** : {is a pandas dataframe} The dataframe that is used to train the model (y variables).

- **title:** {is a String} The title of the model
- **description:** {is a String} The description of the model
- **algorithm:** {is a String} The algorithm that the model implements string

if y is empty generate an empty dataframe with the title of the predicted feature.

The id of the model is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

## 8. `deploy_tree()`

Allows deployment of tree models that are created from scikit-learn algorithms:

- `tree.DecisionTreeClassifier()`
- `tree.DecisionTreeRegressor()`
- `tree.ExtraTreeClassifier()`
- `tree.ExtraTreeRegressor()`

`jaqpot.deploy_tree()` parameters are:

- **model** : {is a sklearn trained model} is a trained model that occurs from the `sklearn.linear_model` family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).
- **y** : {is a pandas dataframe} The dataframe that is used to train the model (y variables).
- **title:** {is a String} The title of the model
- **description:** {is a String} The description of the model
- **algorithm:** {is a String} The algorithm that the model implements string

The id of the model is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

## 9. `deploy_pipeline()`

Allows deployment of pipelined models that are created from scikit-learn algorithms.

`jaqpot.deploy_pipeline()` parameters are:

- **pipeline** : sklearn pipeline model is a trained model that occurs from the `sklearn.linear_model` family of algorithms
- **X** : {is a pandas dataframe} The dataframe that is used to train the model (X variables).
- **y** : {is a pandas dataframe} The dataframe that is used to train the model (y variables).
- **title:** {is a String} The title of the model
- **description:** {is a String} The description of the model
- **algorithm:** {is a String} The algorithm that the model implements string

The id of the model / pipeline is returned. The model can be found on the Jaqpot homepage of the user for editing / sharing / execution (create predictions).

*Example usage*



```
from jaqpotpy import Jaqpot
import pandas as pd
from sklearn import linear_model

df = pd.read_csv('/path/train.csv')
X = df[['Pclass', 'SibSp', 'Parch', 'Fare']]
y = df['Survived']

clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').fit(X, y)

jaqpot.deploy_linear_model(clf, X, y, title="Sklearn 2", description="Logistic regression model from python for the titanic
dataset",
    algorithm="logistic regression")
```

On the above example a linear model (in our case a Logistic Regression model) is created and deployed on jaqpot. The dataset is read as a pandas dataframe (having the variables used for training as pandas dataframes is a requirement for all algorithms in Jaqpot 5) and the X and y dataframes are created, on which the algorithm is trained and the model is created.

## Support

