/Users/allenayodeji/Desktop/DataSets/Cab_Flask_Setup.py

temp.py | Cab_Model.py | Cab_Flask_Setup.py | untitled1.py*

```python
import pickle
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression


def load_pickles(model_pickle_path, label_encoder_pickle_path):
    """
    Loading pickled model and label encoder from the training stage
    """
    model_pickle_opener = open(model_pickle_path,"rb")
    model = pickle.load(model_pickle_opener)

    label_encoder_pickle_opener = open(label_encoder_pickle_path,"rb")
    label_encoder_dict = pickle.load(label_encoder_pickle_opener)

    return model, label_encoder_dict


def pre_process_data(df, label_encoder_dict):
    """
    Converting all non-numeric columns to numeric, using the saved
    encoder from the training stage.
    """
    df.drop("Customer_ID", axis=1, inplace=True)
    df.drop("Transaction_ID", axis=1, inplace=True)
    df.drop("Payment_Mode", axis=1, inplace=True)
    for col in df.columns:
        if col in list(label_encoder_dict.keys()):
            column_le = label_encoder_dict[col]
            df.loc[:, col] = column_le.transform(df.loc[:, col])
        else:
            continue
    return df


def make_predictions(processed_df, model):
    """
    Generating the 2 parts needed for prediction: binary prediction
    (cab[1] or no cab [0]), and saivng as a JSON.
    Probability returned by "predict_proba" method contains 2 probabilities,
    for both neg and positive classes. Returning only probability of positive
    class, cab.
    """
    prediction = model.predict(processed_df)
    probability = model.predict_proba(processed_df)
    probabilities = []
    for prob_array in probability:
        # getting only positive class probability for each one
        probabilities.append(prob_array[1])

    # packaging the predictions into a DF
    predictions_df = pd.DataFrame({"prediction": prediction,
                                   "probability": probabilities})
    # converting predictions DF to json
    predictions_json = predictions_df.to_json(orient="records")
    return predictions_json
```

/Users/allenayodeji/Desktop/DataSets

Name	Type	Size	Value
date_object	date	1	2018-01-04
datetime_date	datetime	1	2018-01-04 00:00:00
df	DataFrame	(359392, 13)	Column names: T
holdout	DataFrame	(3000, 13)	Column names: T
i	int	1	43104
model	linear_model._logistic.LogisticRegression	1	LogisticRegress
pickler	BufferedWriter	1	BufferedWriter
predictions	Array of int64	(3000,)	[1685 2422 2161
processed_train	DataFrame	(3000, 10)	Column names: C
profit	Series	(359392,)	Series object c

Help | Variable Explorer | Plots | Files

Console 1/A

```
   ...:     df.drop("Payment_Mode", axis=1, inplace=True)
   ...:     for col in df.columns:
   ...:         if col in list(label_encoder_dict.keys()):
   ...:             column_le = label_encoder_dict[col]
   ...:             df.loc[:, col] = column_le.transform(df.loc[:, col])
   ...:         else:
   ...:             continue
   ...:     return df

In [53]: def make_predictions(processed_df, model):
   ...:     """
   ...:     Apply saved model to get predictions and probabilities
   ...:     """
   ...:     prediction = model.predict(processed_df)
   ...:     probability = model.predict_proba(processed_df)
   ...:     return prediction, probability

In [54]: if __name__ == "__main__":
   ...:     app.run()
 * Serving Flask app "untitled1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
```

IPython console | History

LSP Python: ready | conda: base (Python 3.9.7) | Line 186, Col 14 | UTF-8 | LF | RW | Mem 66%

/Users/allenayodeji/Desktop/DataSets/Cab_Flask_Setup.py

temp.py | Cab_Model.py | Cab_Flask_Setup.py | untitled1.py*

```python
def generate_predictions():
    """
    Master-function for applying all steps needed to generated predictions.
    """
    # reading in the test JSON of holdout data, which was set aside in training
    test_df = pd.read_json("holdout_test.json")

    # paths to saved pickles
    model_pickle_path = "Cab_prediction_model.pkl"
    label_encoder_pickle_path = "Cab_prediction_label_encoders.pkl"

    model, label_encoder_dict = load_pickles(model_pickle_path,
                                             label_encoder_pickle_path)

    processed_df = pre_process_data(test_df, label_encoder_dict)
    prediction_json = make_predictions(processed_df, model)
    return prediction_json

if __name__ == '__main__':
    prediction_json = generate_predictions()
```

```python
from flask import Flask, request
import numpy as np
import pandas as pd
import pickle

app = Flask(__name__)

model = pickle.load(open('model.pkl','rb'))

@app.route('/predict', methods=['POST'])


def generate_predictions():
    """
    Master-function for applying all steps needed to generated predictions.
    """
    # pulling the input json out of the request
    input = request.json
    # converting input json to DF
    df = pd.DataFrame(input, index=np.arange(len(input)))

    # defining path to pickled model and transformer
    model_pickle_path = "churn_prediction_model.pkl"
```

/Users/allenayodeji/Desktop/DataSets

Name	Type	Size	Value
date_object	date	1	2018-01-04
datetime_date	datetime	1	2018-01-04 00:00:00
df	DataFrame	(359392, 13)	Column names: T
holdout	DataFrame	(3000, 13)	Column names: T
i	int	1	43104
model	linear_model._logistic.LogisticRegression	1	LogisticRegress
pickler	BufferedWriter	1	BufferedWriter
predictions	Array of int64	(3000,)	[1685 2422 2161
processed_train	DataFrame	(3000, 10)	Column names: C
profit	Series	(359392,)	Series object c

Help | Variable Explorer | Plots | Files

Console 1/A

```
...:     df.drop("Payment_Mode", axis=1, inplace=True)
...:     for col in df.columns:
...:         if col in list(label_encoder_dict.keys()):
...:             column_le = label_encoder_dict[col]
...:             df.loc[:, col] = column_le.transform(df.loc[:, col])
...:         else:
...:             continue
...:     return df

In [53]: def make_predictions(processed_df, model):
...:     """
...:     Apply saved model to get predictions and probailities
...:     """
...:     prediction = model.predict(processed_df)
...:     probability = model.predict_proba(processed_df)
...:     return prediction, probability

In [54]: if __name__ == "__main__":
...:     app.run()
 * Serving Flask app "untitled1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
```

IPython console | History

LSP Python: ready | conda: base (Python 3.9.7) | Line 186, Col 14 | UTF-8 | LF | RW | Mem 67%

/Users/allenayodeji/Desktop/DataSets/Cab_Flask_Setup.py

temp.py | Cab_Model.py | Cab_Flask_Setup.py | untitled1.py*

```python
def generate_predictions():
    """
    Master-function for applying all steps needed to generated predictions.
    """
    # pulling the input json out of the request
    input = request.json
    # converting input json to DF
    df = pd.DataFrame(input, index=np.arange(len(input)))

    # defining path to pickled model and transformer
    model_pickle_path = "churn_prediction_model.pkl"
    label_encoder_pickle_path = "churn_prediction_label_encoders.pkl"

    model, label_encoder_dict = load_pickles(model_pickle_path,
                                             label_encoder_pickle_path)
    # calling pre-processing functions
    processed_df = pre_process_data(df, label_encoder_dict)
    # calling prediction funcitons
    prediction, probability = make_predictions(processed_df, model)
    probabilities = []
    for prob_array in probability:
        # getting only positive class probability for each one
        probabilities.append(prob_array[1])

    # packaging the predictions into a DF
    predictions_df = pd.DataFrame({"prediction": prediction,
                                   "probability": probabilities})
    # converting predictions DF to json
    predictions_json = predictions_df.to_json(orient="records")
    return predictions_json


def load_pickles(model_pickle_path, label_encoder_pickle_path):
    """
    Loading pickled model and label encoder from the training stage
    """
    model_pickle_opener = open(model_pickle_path,"rb")
    model = pickle.load(model_pickle_opener)

    label_encoder_pickle_opener = open(label_encoder_pickle_path, "rb")
    label_encoder_dict = pickle.load(label_encoder_pickle_opener)

    return model, label_encoder_dict


def pre_process_data(df, label_encoder_dict):
    """
    Converting all non-numeric columns to numeric, using the saved
    encoder from the training stage.
    """
    df.drop("customerID", axis=1, inplace=True)
    df.drop("Transaction_ID", axis=1, inplace=True)
    df.drop("Payment_Mode", axis=1, inplace=True)
    for col in df.columns:
        if col in list(label_encoder_dict.keys()):
            column_le = label_encoder_dict[col]
            df.loc[:, col] = column_le.transform(df.loc[:, col])
```

/Users/allenayodeji/Desktop/DataSets

Name	Type	Size	Value
date_object	date	1	2018-01-04
datetime_date	datetime	1	2018-01-04 00:00:00
df	DataFrame	(359392, 13)	Column names: T
holdout	DataFrame	(3000, 13)	Column names: T
i	int	1	43104
model	linear_model._logistic.LogisticRegression	1	LogisticRegress
pickler	BufferedWriter	1	BufferedWriter
predictions	Array of int64	(3000,)	[1685 2422 2161
processed_train	DataFrame	(3000, 10)	Column names: C
profit	Series	(359392,)	Series object c
train	DataFrame	(3000, 10)	Column names: C

Help | Variable Explorer | Plots | Files

Console 1/A

```
...:     df.drop("Transaction_ID", axis=1, inplace=True)
...:     df.drop("Payment_Mode", axis=1, inplace=True)
...:     for col in df.columns:
...:         if col in list(label_encoder_dict.keys()):
...:             column_le = label_encoder_dict[col]
...:             df.loc[:, col] = column_le.transform(df.loc[:, col])
...:         else:
...:             continue
...:     return df

In [53]: def make_predictions(processed_df, model):
...:     """
...:     Apply saved model to get predictions and probailities
...:     """
...:     prediction = model.predict(processed_df)
...:     probability = model.predict_proba(processed_df)
...:     return prediction, probability

In [54]: if __name__ == "__main__":
...:     app.run()
 * Serving Flask app "untitled1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
```

IPython console | History

LSP Python: ready | conda: base (Python 3.9.7) | Line 186, Col 14 | UTF-8 | LF | RW | Mem 68%

/Users/allenayodeji/Desktop/DataSets/Cab_Flask_Setup.py

temp.py | Cab_Model.py | Cab_Flask_Setup.py | untitled1.py*

```python
    processed_df = pre_process_data(df, label_encoder_dict)
    # calling prediction funcitons
    prediction, probability = make_predictions(processed_df, model)
    probabilities = []
    for prob_array in probability:
        # getting only positive class probability for each one
        probabilities.append(prob_array[1])

    # packaging the predictions into a DF
    predictions_df = pd.DataFrame({"prediction": prediction,
                                   "probability": probabilities})
    # converting predictions DF to json
    predictions_json = predictions_df.to_json(orient="records")
    return predictions_json


def load_pickles(model_pickle_path, label_encoder_pickle_path):
    """
    Loading pickled model and label encoder from the training stage
    """
    model_pickle_opener = open(model_pickle_path,"rb")
    model = pickle.load(model_pickle_opener)

    label_encoder_pickle_opener = open(label_encoder_pickle_path, "rb")
    label_encoder_dict = pickle.load(label_encoder_pickle_opener)

    return model, label_encoder_dict


def pre_process_data(df, label_encoder_dict):
    """
    Converting all non-numeric columns to numeric, using the saved
    encoder from the training stage.
    """
    df.drop("customerID", axis=1, inplace=True)
    df.drop("Transaction_ID", axis=1, inplace=True)
    df.drop("Payment_Mode", axis=1, inplace=True)
    for col in df.columns:
        if col in list(label_encoder_dict.keys()):
            column_le = label_encoder_dict[col]
            df.loc[:, col] = column_le.transform(df.loc[:, col])
        else:
            continue
    return df


def make_predictions(processed_df, model):
    """
    Apply saved model to get predictions and probabilities
    """
    prediction = model.predict(processed_df)
    probability = model.predict_proba(processed_df)
    return prediction, probability


if __name__ == "__main__":
    app.run()
```

/Users/allenayodeji/Desktop/DataSets

Name	Type	Size	Value
date_object	date	1	2018-01-04
datetime_date	datetime	1	2018-01-04 00:00:00
df	DataFrame	(359392, 13)	Column names: T
holdout	DataFrame	(3000, 13)	Column names: T
i	int	1	43104
model	linear_model._logistic.LogisticRegression	1	LogisticRegress
pickler	BufferedWriter	1	BufferedWriter
predictions	Array of int64	(3000,)	[1685 2422 2161
processed_train	DataFrame	(3000, 10)	Column names: C
profit	Series	(359392,)	Series object c
train	DataFrame	(3000, 10)	Column names: C

Help | Variable Explorer | Plots | Files

Console 1/A

```
...:     df.drop("Transaction_ID", axis=1, inplace=True)
...:     df.drop("Payment_Mode", axis=1, inplace=True)
...:     for col in df.columns:
...:         if col in list(label_encoder_dict.keys()):
...:             column_le = label_encoder_dict[col]
...:             df.loc[:, col] = column_le.transform(df.loc[:, col])
...:         else:
...:             continue
...:     return df

In [53]: def make_predictions(processed_df, model):
...:     """
...:     Apply saved model to get predictions and probabilities
...:     """
...:     prediction = model.predict(processed_df)
...:     probability = model.predict_proba(processed_df)
...:     return prediction, probability

In [54]: if __name__ == "__main__":
...:     app.run()
 * Serving Flask app "untitled1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
```

IPython console | History

LSP Python: ready | conda: base (Python 3.9.7) | Line 182, Col 38 | UTF-8 | LF | RW | Mem 67%