TO-DO-LIST PROGETTO
– FASE 1 –

1. **Documentazione:** Solitamente, chi produce un componente produce pure la documentazione correlata a quel componente.

2. **Estrazione dei Dati:**

   - Creare una Classe **“JSON_Saver”** che avrà un metodo **“save”** che estrapola determinati tipi di candele (volendo anche un solo tipo) di un determinato market (servendosi della classe Extractor) in un determinato intervallo di tempo (aiutarsi con la classe Time_Stamp_Converter per il timestamp) e le salva in una serie di file .json diversificati per tipo di candela.

3. **Refining dei Dati:**

   - Creare una classe **“Dataset_Maker”** che avrà un metodo **extract** che prende in input il path di un file json ,che conterrà candele dello stesso tipo, estratte sfruttando la classe **“JSON_Saver”**.

     A questo punto, il metodo (partendo dal json, che poi sarà un dizionario) dovrà creare esattamente 1 matrice e 1 Vettore:

     ↔ La Matrice è quella delle Osservazioni (contenente tutti i valori tranne quelli che appartengono al Campo ClosePrice) a cui è stato sottratto l’ultimo campo.
     (in sostanza, nella Matrice restano i seguenti campi:
     CloseTime, OpenPrice, HighPrice, LowPrice, Volume).

     ↔ Il Vettore è il Vettore (Colonna) delle Etichette (Contenente esclusivamente tutti i valori del Campo ClosePrice).

     A questo punto, il compito del metodo **extract** finisce scrivendo in un file .pkl (sfruttando il modulo Python “cPickle”) la Matrice delle Osservazioni e il Vettore delle Etichette.

     Nella buona sostanza, con **extract** abbiamo creato un Dataset.

   - Infine, creare una classe “**Dataset_Creator”** che ingloba tutti i metodi delle seguenti classi:

     - Extractor
     - Time_Stamp_Converter
     - Periods_Maker
     - JSON_Saver
     - Dataset_Maker

4. **Cleaning dei Dati:**

   - Creare una classe **“Cleaner”** che ha due metodi: features_scaling (si commenta da solo) e mean_normalization (si commenta da solo).

5. **Learnig:**

   - Creare una classe **“Dataset_Splitter”** che ha un metodo **“split”** che (a partire da un file .pkl di un tipo di candele creato con la classe **“Datase_Creator”**) si occupa ,in prima battuta, di suddividere il Dataset in Training-Set (70 %) e Test-Set (30 %).

     In seconda battuta, questo metodo dovrà scrivere in 2 file .pkl rispettivamente: il Training-Set e il Test-Set.

   - Creare una classe **Curve** che ha metodi che disegnano rispettivamente i seguenti Grafici:

     1. Grafico Ipotesi (2D)
     2. Grafico Ipotesi (3D)
     3. Grafico Grado del Polinomio VS Errore (2D)
     4. Grafico Lambda VS Error (2D) **(IN DUBBIO)**
     5. Grafico Training-Set-Size VS Error (2D) **(IN DUBBIO)**
     6. REC Curve **(IN DUBBIO)**

   - Creare una classe **“Dataset_Projector”** che ha un metodo **to_project (X_Training, list_of_projected_features)**, che restringe la Matrice delle osservazioni in Input, in particolar modo saranno mantenute nella matrice tutte le features contenute nella lista passata in input.
     Tutto questo serve per Rappresentare un Modello.
     Inoltre questo metodo ,oltre a restringere la Matrice delle Osservazioni, può anche elevare a potenza tutti gli elementi ,contenuti nella matrice, appartenenti ad una o più feature.
     Tutto questo serve per rappresentare un Modello di Ordine Superiore.

     PS: **list_of_projected_features** è una lista, dove ogni elemento della lista è un numero Maggiore di 0 oppure 0; dove il Valore Maggiore di 0 esprime il fatto che la feature è presente in quel modello, inoltre il Valore maggiore di 0 ci dice anche il Grado delle Feature in quel modello.
     Invece il Valore 0 esprime il fatto che quella feature NON e’ presente nel modello.
     Praticamente **list_of_projected_features** è la rappresentazione di un modello.

   - Creare una classe **Model_Builder** che ha un metodo **build** che costruisce la lista di liste che si passa al metodo **train_models** della classe **Trainer**.
     In sostanza la struttura restituita da **build** rappresenta una lista di Modelli, dove ogni elemento della Lista è una struttura del tutto identica a quelle che si passano in input al metodo **to_project** della classe **Dataset_Projector**.
     Oltre a ciò, la classe **Model_Builder** deve avere un metodo **print_last_list** che stampa l’ultima lista di Modelli creata; inoltre la classe **Model_Builder** deve avere pure un metodo **save_models** per salvare i Modelli su un file e un metodo **load_models** per caricare da file una lista di Modelli.

- Creare una classe **“Trainer”** che ha un metodo **train_models** che allena una serie di modelli (anche uno solo volendo) sfruttando le Equazioni Normali.

  Inoltre questa classe deve avere un metodo **take_the_best_model_from_last_training** che restituisce una tupla contenente:

  - l’estimatore migliore
  - le features contenute nell’estimatore migliore
  - l’errore J-CV dell’estimatore Migliore
  - l’errore J-Train dell’estimatore Migliore

  Inoltre la classe deve avere un metodo **save_best_estimator** che conserva in 2 file .obj (usando il modulo Pickle) il miglior estimatore e le sue features. (Quest’ultima cosa ci sarà utile in fase di Inferenza).

  Inoltre, la classe **Trainer** deve avere un metodo **print_models_errors** che stampa per ogni modello l’errore J-Train e l’errore J-CV.

  Infine, la classe **Trainer** deve avere un metodo **plot_graphics** che non fa altro che stampare tutti i Grafici delle Ipotesi dei Modelli Allenati (in 2D), inoltre questo metodo stampa pure i Grafici Grado vs Errore nel caso in cui siano presenti ,nell’allenamento, gruppi di Modelli che differiscano solo per il grado.

  In ultima analisi, la classe **Trainer** deve avere un metodo **are_equals** che dice se due modelli sono uguali, uguali tranne per il grado o sono diversi; inoltre la classe **Trainer** deve avere un metodo **take_degree** che dice il grado del modello passato in input.

- Successivamente, creare due classi:

  1. **Recorder_Degree,**
     E’ una classe che serve per tracciare il grafico “Grado del Polinomio VS Errore”.
     Questa classe conterrà tre liste: la lista dei gradi, la lista degli errori di training e la lista degli errori del convalidation-test.
     Ogni volta che si completa una serie di allenamenti dello stesso modello in cui il grado del modello aumenta sempre di più fare le seguenti cose:
     alla fine di ogni allenamento si inserisce il grado del Modello e i valori di J-train e J-cv nelle tre liste (Lista dei Gradi,Lista degli Errori di Training, Lista degli Errori di Convalidation-Test).

  2. **Recorder_Lambda (IN DUBBIO),**
     E’ una classe che serve per tracciare il grafico “Lambda VS Errore”.
     Questa classe conterrà tre liste: la lista dei Lambda, la lista degli errori di training e la lista degli errori di convalidation-test.
     Ogni volta che si completa una serie di allenamenti dello stesso modello in cui il parametro lambda viene sempre cambiato fare le seguenti cose:
     per ogni allenamento completato si inserisce il valore di lambda di quell’allenamento e i valori di J-train e J-cv nelle tre liste (Lista dei Lambda,Lista degli Errori di Training, Lista degli Errori di Convalidation-Test).

6.Valutazione dei Risultati:

- Costruire una classe **“MSE”**, che ha un metodo che calcola l’MSE sul Dataset in input sfruttando un Modello, anch’esso passato in input.

7.Inferenza:

- Creare un piccolo software ,anche a riga di comando, che prende in input **alcuni parametri** (stabiliti nella fase di Learning) di una candela di 12 h non ancora chiusa, e che dia in output la predizione del prezzo di chiusura della Candela, sfruttando la classe Model (opportunamente settata) nel caso della Discesa del Gradiente oppure la classe LinearRegression (opportunamente settata) nel caso di Equazioni Normali.