Demand Forecast

Release v1.0.5

Eduard Poliakov

Jun 12, 2024

CONTENTS:

CHAPTER

ONE

SRC_DEMAND_FORECAST

1.1 src_demand_forecast package

1.1.1 Subpackages

1.1.1.1 src_demand_forecast.data package

Submodules

src_demand_forecast.data.split_dataset module

This module contains the function to split the data into train and test sets.

src_demand_forecast.data.split_dataset.**split_train_test**(*df: DataFrame*, *test_days: int = 30*) → Tuple[DataFrame, DataFrame]

Split the data into train and test sets. The last *test_days* days are held out for testing.

Parameters

- **df** (*pd.DataFrame*) – The input DataFrame containing the data.
- **test_days** (*int*) – The number of days to include in the test set (default: 30). use ">=" sign for df_test

Returns
A tuple containing the train and test DataFrames.

Return type
Tuple[pd.DataFrame, pd.DataFrame]

Module contents

1.1.1.2 src_demand_forecast.download package

Submodules

src_demand_forecast.download.download_from_s3 module

This module is used to download the dataset from the Yandex.Cloud storage.

src_demand_forecast.download.download_from_s3.**download_dataset**(*s3_bucket: str*, *remote_path: str*, *output_path: str*, *file_names: list | None = None*)

Module contents

1.1.1.3 src_demand_forecast.entities package

Submodules

src_demand_forecast.entities.feature_params module

This module contains template the class for feature

class src_demand_forecast.entities.feature_params.**FeatureParams**(*sku_demand_day: List[str]*, *features: Dict[str, Tuple[str, int, str, int | None]]*, *targets: Dict[str, Tuple[str, int]]*)

Bases: object

Class for feature parameters.

features: Dict[str, Tuple[str, int, str, int | None]]

sku_demand_day: List[str]

targets: Dict[str, Tuple[str, int]]

src_demand_forecast.entities.model_params module

This module contains template class for model parameters.

class src_demand_forecast.entities.model_params.**ModelParams**(*features: List[str]*, *horizons: List[int]*, *quantiles: List[float]*)

Bases: object

Class for model parameters.

features: List[str]

horizons: List[int]

quantiles: List[float]

src_demand_forecast.entities.split_params module

This module contains template for split parameters.

class src_demand_forecast.entities.split_params.**SplitParams**(*test_days: int*)

Bases: object

Class for split parameters.

test_days: int

src_demand_forecast.entities.train_pipeline_params module

src_demand_forecast.entities.validation_params module

This module contains template for the validation parameters.

class src_demand_forecast.entities.validation_params.**LowStockSKURequest**(**, confidence_level: float, horizon_days: int, sku_stock: List[SKUInfo]*)

Bases: BaseModel

The LowStockSKURequest scheme for FastAPI.

confidence_level: float

horizon_days: int

sku_stock: List[*SKUInfo*]

class src_demand_forecast.entities.validation_params.**SKUInfo**(**, sku_id: int, stock: int = 0*)

Bases: BaseModel

The SKUInfo scheme for FastAPI.

sku_id: int

stock: int

class src_demand_forecast.entities.validation_params.**SKURequest**(**, sku: SKUInfo, horizon_days: int = 7, confidence_level: float = 0.1*)

Bases: BaseModel

The SKURequest scheme for FastAPI.

confidence_level: float

horizon_days: int

sku: *SKUInfo*

Module contents

1.1.1.4 src_demand_forecast.features package

Submodules

src_demand_forecast.features.AddFeatures module

src_demand_forecast.features.AddTargets module

src_demand_forecast.features.build_sku_by_day module

This module contains functions for creating a DataFrame with sales data.

src_demand_forecast.features.build_sku_by_day.**save_sku_demand_by_day**(*path: str*, *data: DataFrame*)

src_demand_forecast.features.build_sku_by_day.**sku_demand_by_day**(*demand_orders: DataFrame*, *demand_orders_status: DataFrame*) → DataFrame

Converts data from SQL to pandas DataFrame.

1. Converts 'timestamp' into a datetime object and creates a new 'day' column with the date.
2. Creates a cross combination of unique days and unique SKUs.
3. Defines order IDs with delivery statuses (1, 3, 4, 5, 6).
4. Calculates the total number of products sold for each pair (day, SKU).
5. Combines sales data with SKU information.
6. Returns the result with columns 'day', 'sku_id', 'sku', 'price', and 'qty' sorted by 'sku_id' and 'day'.

Parameters

- **demand_orders** (*pd.DataFrame*) – The demand_orders DataFrame.
- **demand_orders_status** (*pd.DataFrame*) – The demand_orders_status DataFrame.

Returns
The DataFrame with the sales data.

Return type
pd.DataFrame

src_demand_forecast.features.build_transformer module

This module contains functions for building a transformer and saving transformed data.

src_demand_forecast.features.build_transformer.**features_and_targets_transformer**() → Pipeline

Builds transformer from config. :returns: The transformer. :rtype: Pipeline

src_demand_forecast.features.build_transformer.**save_transformed_data**(*path: str*, *data: DataFrame*)

Module contents

1.1.1.5 src_demand_forecast.inference package

Submodules

src_demand_forecast.inference.make_request module

This module contains functions for making requests to the FastAPI server.

src_demand_forecast.inference.make_request.**check_response**(*response*) → None

Check the response from the FastAPI server.

Parameters
response (*requests.models.Response*) – The response from the FastAPI server.

Return type
None

src_demand_forecast.inference.make_request.**how_much_to_order**(*sku_id: int*, *stock: int*, *horizon_days: int*, *confidence_level: float*) → None

Make a request to the FastAPI server to get the order quantity.

Parameters

- **sku_id** (*int*) – The SKU id.
- **stock** (*int*) – The current stock level.
- **horizon_days** (*int*) – The number of days in the horizon.
- **confidence_level** (*float*) – The confidence level.

Return type
None

src_demand_forecast.inference.make_request.**low_stock_sku_list**(*confidence_level: float*, *horizon_days: int*, *sku_stock: List[dict]*) → None

Get the list of low stock SKUs.

Parameters

- **confidence_level** (*float*) – The confidence level.
- **horizon_days** (*int*) – The number of days in the horizon.
- **sku_stock** (*List[dict]*) – The list of dictionaries with the SKU id and stock level.

Return type
None

src_demand_forecast.inference.make_request.**stock_level_forecast**(*sku_id: int*, *stock: int*, *horizon_days: int*, *confidence_level: float*) → None

Get the stock level forecast.

Parameters

- **sku_id** (*int*) – The SKU id.
- **stock** (*int*) – The current stock level.

- **horizon_days** (*int*) – The number of days in the horizon.
- **confidence_level** (*float*) – The confidence level.

Return type
None

Module contents

1.1.1.6 src_demand_forecast.models package

Submodules

src_demand_forecast.models.repro_experiments module

This module contains the MultiTargetModel class, which is used to train a quantile regression model for multiple targets.

class src_demand_forecast.models.repro_experiments.**MultiTargetModel**(*features: List[str]*, *horizons: List[int] = [7, 14, 21]*, *quantiles: List[float] = [0.1, 0.5, 0.9]*)

Bases: object

fit(*data: DataFrame*, *verbose: bool = False*) → None

Fit model on data.

Parameters

- **data** (*pd.DataFrame*) – Data to fit on.
- **verbose** (*bool, optional*) – Whether to show progress bar, by default False Optional to implement, not used in grading.

Return type
None

predict(*data: DataFrame*) → DataFrame

Predict on data. Predict 0 values for a new sku_id. :param data: Data to predict on. :type data: pd.DataFrame

Returns
Predictions.

Return type
pd.DataFrame

src_demand_forecast.models.repro_experiments.**evaluate_model**(*df_true: DataFrame*, *df_pred: DataFrame*, *quantiles: List[float] = [0.1, 0.5, 0.9]*, *horizons: List[int] = [7, 14, 21]*) → DataFrame

Evaluate model on data.

Parameters

- **df_true** (*pd.DataFrame*) – True values.
- **df_pred** (*pd.DataFrame*) – Predicted values.

- **quantiles** (*List[float], optional*) – Quantiles to evaluate on, by default [0.1, 0.5, 0.9].
- **horizons** (*List[int], optional*) – Horizons to evaluate on, by default [7, 14, 21].

Returns
Evaluation results.

Return type
pd.DataFrame

src_demand_forecast.models.repro_experiments.**quantile_loss**(*y_true: ndarray*, *y_pred: ndarray*, *quantile: float*) → float

Calculate the quantile loss between the true and predicted values.

The quantile loss measures the deviation between the true
and predicted values at a specific quantile.

Parameters

- **y_true** (*np.ndarray*) – The true values.
- **y_pred** (*np.ndarray*) – The predicted values.
- **quantile** (*float*) – The quantile to calculate the loss for.

Returns
The quantile loss.

Return type
float

src_demand_forecast.models.repro_experiments.**serialize_model**(*model*, *output: str*) → None

Serialize model to pickle file.

Parameters

- **model** (*object*) – Model to serialize.
- **output** (*str*) – Path to output file.

Returns
Path to output file.

Return type
str

src_demand_forecast.models.train_model module

This module contains the code to train a multi-target model. The model is trained on the data and then used to make predictions. The predictions are evaluated using the quantile loss metric.

class src_demand_forecast.models.train_model.**MultiTargetModel**(*features: List[str]*, *horizons: List[int] = [7, 14, 21]*, *quantiles: List[float] = [0.1, 0.5, 0.9]*)

Bases: object

fit(*data: DataFrame*, *verbose: bool = False*) → None

Fit model on data.

Parameters

- **data** (*pd.DataFrame*) – Data to fit on.
- **verbose** (*bool, optional*) – Whether to show progress bar, by default False Optional to implement, not used in grading.

predict(*data: DataFrame*) → DataFrame

Predict on data. Predict 0 values for a new sku_id. :param data: Data to predict on. :type data: pd.DataFrame

Returns
Predictions.

Return type
pd.DataFrame

src_demand_forecast.models.train_model.**evaluate_model**(*df_true: DataFrame*, *df_pred: DataFrame*, *quantiles: List[float] = [0.1, 0.5, 0.9]*, *horizons: List[int] = [7, 14, 21]*) → DataFrame

Evaluate model on data.

Parameters

- **df_true** (*pd.DataFrame*) – True values.
- **df_pred** (*pd.DataFrame*) – Predicted values.
- **quantiles** (*List[float], optional*) – Quantiles to evaluate on, by default [0.1, 0.5, 0.9].
- **horizons** (*List[int], optional*) – Horizons to evaluate on, by default [7, 14, 21].

Returns
Evaluation results.

Return type
pd.DataFrame

src_demand_forecast.models.train_model.**quantile_loss**(*y_true: ndarray*, *y_pred: ndarray*, *quantile: float*) → float

Calculate the quantile loss between the true and predicted values.

The quantile loss measures the deviation between the true
and predicted values at a specific quantile.

Parameters

- **y_true** (*np.ndarray*) – The true values.
- **y_pred** (*np.ndarray*) – The predicted values.
- **quantile** (*float*) – The quantile to calculate the loss for.

Returns
The quantile loss.

Return type
float

src_demand_forecast.models.train_model.**serialize_model**(*model*, *output: str*) → None

Serialize model to pickle file.

Parameters

- **model** (*object*) – Model to serialize.
- **output** (*str*) – Path to output file.

Module contents

1.1.1.7 src_demand_forecast.upload package

Submodules

src_demand_forecast.upload.s3_storage module

This script uploads files to Yandex Object Storage (Yandex S3).

src_demand_forecast.upload.s3_storage.**upload_dataset**(*local_path: str*, *s3_bucket: str*, *remote_path: str*, *file_names: list | None = None*)

Module contents

1.1.1.8 src_demand_forecast.visualization package

Submodules

src_demand_forecast.visualization.visualize module

In this file, we will implement the visualization of the demand forecast.

Module contents

1.1.2 Module contents

CHAPTER

TWO

DEMAND FORECAST

2.1 What is the project about?

Products such as electronics, household appliances have varying characteristics and demand cycles. Category managers, responsible for overseeing these items throughout their lifecycle, face challenges in planning purchases, especially when the assortment spans thousands of items. As data volume increases and platform complexity grows, traditional methods of inventory management and demand forecasting become less effective. In response to these challenges, Supermegaretaillite is investing in the development of an automated system, including an ML-based demand forecasting service.

To preserve the primacy of the source code, the SRC module is loaded in PyPi for further use in repositories training-pipeline and inference-pipeline.

2.2 How to use this project?

This project is deployed on the server and is fully ready to work 24/7. If you want to use this service, you need to: 1. Open this web address. 2. For example, you would like to know the demand for the product with SKU ID 20 for the next 7 days. You should fill in the fields in web service: SKU_ID = 20, Stock = 10, Horizon Days = 7, Confidence Level = 0.90. 3. Click the buttons: - “Get how much to order” to find out how much inventory you need to order from the supplier. - “Get stock level” to find out how much stock you will have in 7 days. - “Get low stock SKU ID” to find out which products will be out of stock in 7 days.

2.3 Project Organization

```
.
├── app.py                <--- FastAPI app
├── configs               <--- Configs for this project
│   └── train_config.yaml     <--- Training configuration file
├── data                  <--- Data for this project
│   ├── external              <--- External data sources
│   ├── interim               <--- Intermediate data
│   ├── processed             <--- Processed data ready for analysis
│   │   ├── features_targets.csv  <--- Features and targets for training
│   │   ├── predictions.csv     <--- Model predictions
│   │   └── sku_demand_day.csv  <--- Demand forecast for each SKU
│   └── raw                   <--- Raw data before processing
│       ├── demand_orders.csv  <--- Demand orders data
```

(continues on next page)

(continued from previous page)

```
│       ├── demand_orders_status.csv  <--- Demand orders status data
│       ├── features.csv     <--- Features data
│       └── sales.csv       <--- Sales data
├── data.dvc               <--- DVC data tracking file
├── docker-compose.yml     <--- Docker Compose configuration
├── Dockerfile             <--- Dockerfile for building the project container
├── docs                   <--- Documentation for this project
│   ├── Makefile               <--- Makefile for building docs
│   └── ML_System_Design.md    <--- ML System Design document
├── dvc.lock               <--- DVC lock file
├── dvc.yaml               <--- DVC pipeline definition
├── images                 <--- Images used in documentation
│   └── Demand_Forencast_Pipeline.jpg
├── __init__.py            <--- Package initializer
├── LICENSE                <--- License for this project
├── logs                   <--- Logs generated by the application
│   ├── app.log
├── Makefile               <--- Makefile for various build tasks
├── MANIFEST.in            <--- Manifest for package distribution
├── models                 <--- ML models and associated files
│   ├── losses.json            <--- Model training losses
│   └── model.pkl              <--- Serialized model
├── notebooks              <--- Jupyter notebooks for exploration and analysis
│   └── EDA.ipynb              <--- Exploratory Data Analysis notebook
├── project_structure.txt  <--- Project structure description
├── prometheus_data        <--- Prometheus monitoring configuration
│   ├── app_metrics.py         <--- Application metrics for Prometheus
│   └── prometheus.yml         <--- Prometheus configuration file
├── README.md              <--- Readme file with project overview
├── requirements.txt       <--- Python dependencies
├── setup.cfg              <--- Setup configuration
├── setup.py               <--- Setup script for installing the package
├── src_demand_forecast    <--- Source code for demand forecast
│   ├── data                   <--- Data processing scripts
│   │   ├── __init__.py
│   │   └── split_dataset.py   <--- Script for splitting the dataset
│   ├── download               <--- Data download scripts
│   │   ├── download_from_s3.py<--- Script to download data from S3
│   │   └── __init__.py
│   ├── entities               <--- Entity definitions
│   │   ├── feature_params.py
│   │   ├── __init__.py
│   │   ├── model_params.py  <--- Model parameters
│   │   ├── split_params.py  <--- Split parameters
│   │   ├── train_pipeline_params.py  <--- Training pipeline parameters
│   │   └── validation_params.py  <--- Validation parameters
│   ├── features               <--- Feature engineering scripts
│   │   ├── AddFeatures.py  <--- Script for adding features
│   │   ├── AddTargets.py  <--- Script for adding targets
│   │   ├── build_sku_by_day.py  <--- Script for building SKU demand by day
│   │   ├── build_transformer.py  <--- Script for building a transformer
│   │   └── __init__.py
```

(continues on next page)

(continued from previous page)

```
│   ├── inference              <--- Inference scripts
│   │   ├── __init__.py
│   │   └── make_request.py    <--- Script for making inference requests
│   ├── __init__.py
│   ├── models                 <--- Model training and evaluation scripts
│   │   ├── __init__.py
│   │   ├── repro_experiments.py <--- Reproducibility experiments
│   │   └── train_model.py     <--- Model training script
│   ├── upload                 <--- Data upload scripts
│   │   ├── __init__.py
│   │   └── s3_storage.py      <--- Script to upload data to S3
│   └── visualization          <--- Data visualization scripts
│       ├── __init__.py
│       └── visualize.py       <--- Script for visualizing data
├── tests                   <--- Unit tests for the project
│   ├── app                    <--- Tests for the FastAPI app
│   │   └── test_streamlit_app.py  <--- Tests for the Streamlit app
│   ├── conftest.py          <--- Pytest configuration
│   ├── data                   <--- Tests for data processing
│   │   └── test_data.py     <--- Tests for data processing
│   └── models                 <--- Tests for models
├── tox.ini                 <--- Tox configuration for testing
├── train_pipeline.py       <--- Script for running the training pipeline
└── web_app                 <--- Web application scripts
    ├── Dockerfile          <--- Dockerfile for the web application
    ├── __init__.py
    ├── requirements.txt  <--- Python dependencies for the web application
    └── streamlit_app.py    <--- Streamlit web application
```

PYTHON MODULE INDEX

S