Project Functions Documentation

Table of contents

Overview

This document describes the public functions in the project, covering their purpose, parameters, return values, side effects, usage examples, and implementation notes. The functions are defined across the following modules: `read.py`, `preprocess.py`, `model.py`, and `viz.py`.

Module: `read.py`

`get_data_path(filename: str)`

Purpose: Return a path-like object pointing to `data/<filename>` within the package using `importlib.resources`.

Parameters: - `filename` *(str)*: File name, e.g., `"sales.csv"`.

Returns: - Path-like object to `data/<filename>`.

Example:

```
from read import get_data_path
csv_path = get_data_path("vgsales.csv")
```

`read_data(file_path)`

Purpose: Read CSV into a DataFrame; convert `Year` to pandas nullable integer (`Int64`); drop `"Unnamed: 0"` column; return cleaned DataFrame.

Parameters: - `file_path`: String or path-like to the CSV.

Returns: - `sales` *(pd.DataFrame)*: Cleaned dataset.

Side effects: Changes `Year` dtype; drops `"Unnamed: 0"` if present.

Example:

```
from read import read_data, get_data_path
sales = read_data(get_data_path("sales.csv"))
```

Module: `preprocess.py`

`process_data(sales)`

Purpose: Deduplicate and aggregate raw sales records to the `Name` level, collecting distinct categorical lists and summarizing numeric metrics.

Aggregations: - `Platform`: list of unique non-null platforms - `Year`: max - `Genre`: list of unique non-null genres - `Publisher`: list of unique non-null publishers - `all_time_peak`: max - `last_30_day_avg`: max - `NA_Sales`, `EU_Sales`, `JP_Sales`, `Other_Sales`, `Global_Sales`: sum

Returns: - `sales_combined` *(pd.DataFrame)* with one row per `Name`.

Example:

```
from preprocess import process_data
sales_combined = process_data(sales)
```

`prepare_data(sales_combined)`

Purpose: Ensure list types for `Platform`/`Genre`; multi-hot encode them via `MultiLabelBinarizer`; concatenate encoded features with selected numeric predictors; drop rows with missing `Year`.

Returns: - `final_df` *(pd.DataFrame)* containing: - Numeric predictors: `all_time_peak`, `last_30_day_avg`, `Year`, `Rank` - Target(s): includes `Global_Sales` - Encoded features: `Platform_<category>`, `Genre_<category>`

Example:

```
from preprocess import prepare_data
final_df = prepare_data(sales_combined)
```

Module: `model.py`

`rf_fit(final_df, area)`

Purpose: Train a `RandomForestRegressor` to predict `area` using 3-fold `GridSearchCV`. The function splits the data, scales numeric features on the training set, logs the target via `np.log1p`, evaluates on a held-out test set, prints metrics and feature importances, and returns the trained estimator together with the fitted `StandardScaler` used for numeric features.

Key steps: - Split: `train_test_split(test_size=0.2, random_state=42)` - Scale: `StandardScaler` is fit on the training set for the numeric features `['all_time_peak', 'last_30_day_avg', 'Year']` (the scaler is returned so it can be reused when preparing new data for prediction). - Grid: `n_estimators=[50,100,200]`, `max_depth=[5,10,15]`, `min_samples_split=[2,3,5]`, `min_samples_leaf=[1,2,3]`

Returns: - `(best_model, scaler)` — `best_model` is the `RandomForestRegressor` selected by grid search, and `scaler` is the fitted `StandardScaler` object used during training.

Outputs (printed): Best parameters, R^2, RMSE (log scale), top-10 feature importances.

Example:

```
from model import rf_fit
best_model, scaler = rf_fit(final_df, area="Global_Sales")
```

`predict(best_model, area, new_data, scaler=None)`

Purpose: Prepare `new_data` using the same numeric scaling applied during training, predict with `best_model`, and inverse the log transform via `np.expm1` to obtain predictions on the original scale.

Parameters: - `best_model`: fitted estimator from `rf_fit` - `area`: target name (not used by the function but kept for API compatibility) - `new_data`: `pd.DataFrame` with the predictor columns used during training - `scaler` *(optional)*: the `StandardScaler` instance returned by `rf_fit`. Passing this scaler is strongly recommended so numeric features are transformed consistently. If `scaler` is `None`, a new `StandardScaler` instance will be created — note that this new scaler will not be fitted and `transform()` will raise an error unless the caller provides already-scaled numeric columns.

Returns: - `preds_exp` *(np.ndarray)*: Predictions in the original target scale (inverse of `log1p`).

Example:

```
from model import predict
# Use the scaler returned from rf_fit for consistent scaling
preds = predict(best_model, area="Global_Sales", new_data=new_X, scaler=scaler)
```

Module: `viz.py`

`print_genre_distribution(sales, genre, area)`

Purpose: Plot a histogram (Seaborn `histplot`) of `area` restricted to rows whose `Genre` contains the given substring. Shows the plot via `plt.show()`.

Parameters: - `sales`: DataFrame of sales records - `genre`: Substring to match within `Genre` (case-sensitive) - `area`: Numeric column to plot (e.g., `Global_Sales`)

Example:

```
from viz import print_genre_distribution
print_genre_distribution(sales, genre="Action", area="Global_Sales")
```

`print_platform_distribution(sales, platform, area)`

Purpose: Plot a histogram of `area` restricted to rows whose `Platform` contains the given substring. Shows the plot via `plt.show()`.

Parameters: - `sales`: DataFrame of sales records - `platform`: Substring to match within `Platform` (case-sensitive) - `area`: Numeric column to plot (e.g., `Global_Sales`)

Example:

```
from viz import print_platform_distribution
print_platform_distribution(sales, platform="PS4", area="Global_Sales")
```