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>`.

Notes: Ensure your package includes a `data/` directory and resources are accessible.

Example:

```
from read import get_data_path
csv_path = get_data_path("sales.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>`

Notes: Adjust which targets/predictors are included based on modeling needs.

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`. Splits data, scales numeric features on the training set, logs target via `np.log1p`, evaluates on test, prints metrics and feature importances, returns best estimator.

Key steps: - Split: `train_test_split(test_size=0.2, random_state=42)` - Scale: `StandardScaler` on `['all_time_peak', 'last_30_day_avg', 'Year', 'Rank']` - Grid: `n_estimators=[50,100,200]`, `max_depth=[5,10,15]`, `min_samples_split=[1,3,5]`, `min_samples_leaf=[1,2,3]` (note: `min_samples_split` must be 2)

Returns: - `best_model` *(RandomForestRegressor)* trained on the full training set via grid search.

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

Implementation notes: - Replace `min_samples_split` values with `[2,3,5]` to comply with scikit-learn requirements. - Consider moving scaling into a `Pipeline` to ensure consistent preprocessing at inference time.

Example:

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

`predict(best_model, area, new_data)`

Purpose: Scale numeric features in `new_data`, 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 in current implementation) - `new_data`: DataFrame with the same predictor columns used during training

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

Implementation caveat: A **new** `StandardScaler()` is created and `transform` is called without fitting to training statistics, which will raise an error and/or cause inconsistency. Persist and reuse the fitted scaler, or use a `Pipeline`.

Example:

```
from model import predict
# Ensure consistent scaling via a Pipeline or by reusing the trained scaler
preds = predict(best_model, area="Global_Sales", new_data=new_X)
```

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")
```

Version & Metadata (`__init__.py`)

- `__version__ = "0.1.5"` identifies the current package version.
- The module re-exports the public API and defines `__all__` to make the import surface explicit.

End-to-End Workflow (Example)

```
from read import get_data_path, read_data
from preprocess import process_data, prepare_data
from model import rf_fit, predict
from viz import print_genre_distribution, print_platform_distribution

# 1) Read raw CSV
sales = read_data(get_data_path("sales.csv"))  # path helper + CSV reader

# 2) Aggregate and prepare features
sales_combined = process_data(sales)
final_df = prepare_data(sales_combined)

# 3) Train a model (predict Global_Sales)
best_model = rf_fit(final_df, area="Global_Sales")
```

```
# 4) Visualize distributions
print_genre_distribution(sales, genre="Action", area="Global_Sales")
print_platform_distribution(sales, platform="PS4", area="Global_Sales")

# 5) Predict on new data (ensure consistent preprocessing)
X_new = final_df.drop(columns=["Global_Sales"]).iloc[:5]
preds = predict(best_model, area="Global_Sales", new_data=X_new)
```

Recommendations

- **Fix hyperparameter grid:** Use `min_samples_split >= 2` to avoid errors.
- **Consistent scaling:** Move scaling into a `Pipeline` or persist the trained scaler for inference.
- **Schema checks:** Validate required columns before processing; log encoder class counts in `prepare_data`.