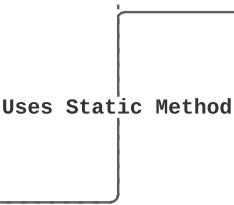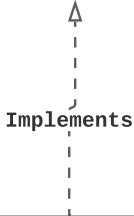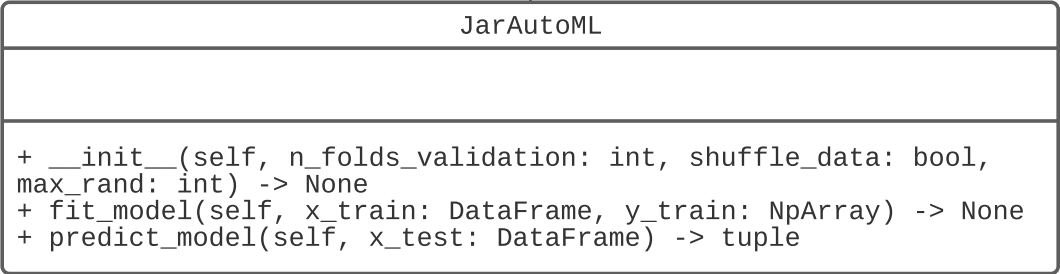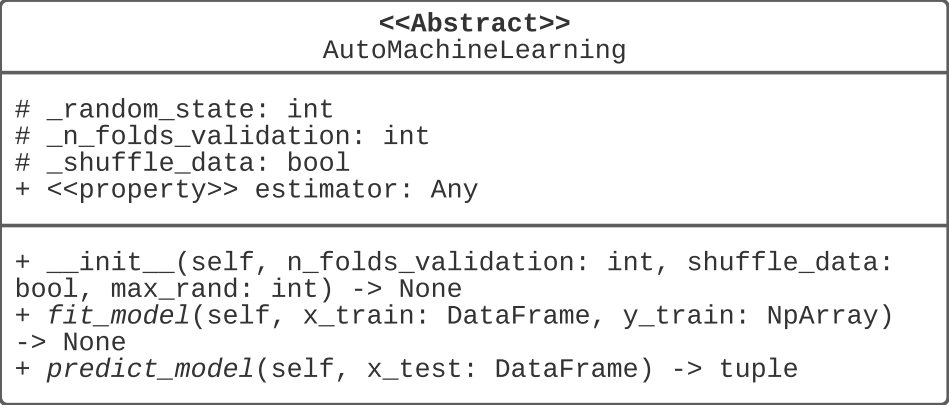```
DataFrame = pd.DataFrame
NpArray = np.ndarray
```

Calls
get_random_number_range
int

**Randomizer**

get_random_number_range_int(start: int, end: int, step: int) -> int

**<<Abstract>>**
**AutoMachineLearning**

\# _random_state: int
\# _n_folds_validation: int
\# _shuffle_data: bool
+ <<property>> estimator: Any

+ __init__(self, n_folds_validation: int, shuffle_data: bool, max_rand: int) -> None
+ *fit_model*(self, x_train: DataFrame, y_train: NpArray) -> None
+ *predict_model*(self, x_test: DataFrame) -> tuple

**Uses Static Method**

**Uses**

**AutoExecutioner**

\# _auto_ml: AutoMachineLearning

+ __init__(self, auto_ml: AutoMachineLearning) -> None
+ train_model(self, df: DataFrame, size : float = 0.0) -> None
+ score_model(self, test: DataFrame, expected: tuple) -> float
+ get_model(self) -> str

**Implements**

**JarAutoML**

+ __init__(self, n_folds_validation: int, shuffle_data: bool, max_rand: int) -> None
+ fit_model(self, x_train: DataFrame, y_train: NpArray) -> None
+ predict_model(self, x_test: DataFrame) -> tuple

**Uses static method**

**SplitterReturner**

+ train_test_split (x: DataFrame, y: NpArray, size: float) -> tuple
+ split_x_y_from_df(df: DataFrame, ravel_data: bool = True) -> tuple