```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder, StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, cross_validate
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier, HistGradientBoostingClassifier
from scipy.stats import loguniform, randint, uniform
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report,
ConfusionMatrixDisplay

RANDOM_STATE = 42

bank = pd.read_csv("bank-full.csv", sep=";")
bank.y = bank.y.map({'yes':1, 'no':0})

[col for col in bank.columns if bank[col].dtype == 'category']

bank.info()

bank.describe()
# include='all'

# drop rows with na
# bank.dropna()

# bank.isna().any()
# bank.isna().sum()
# bank.fillna()

for i in list(bank.columns):
    print(f"{i:<10}->  {bank[i].nunique():<5} unique values")

bank.job.value_counts(ascending=False)

X = bank.drop(columns=["y", 'pdays', 'poutcome', 'day', 'month', 'campaign', 'previous', 'default',
'housing', 'contact'])
y = bank["y"]
X_train, X_test, y_train, y_test = train_test_split(X
                                                    , y
                                                    , test_size=0.3
                                                    , random_state=RANDOM_STATE
                                                    , stratify=y)

X_train['age'].plot(kind = 'hist', bins=77, density=True)


from sklearn.preprocessing import PowerTransformer
log_transformer = PowerTransformer(standardize=False)
pd.DataFrame(log_transformer.fit_transform(X_train[['age']])).plot(kind = 'hist', bins=200, density=True)




num_cols = ['age', 'balance', 'duration']
ord_cols = ['education']
cat_cols = [col for col in X.columns if col not in num_cols and col not in ord_cols]
cat_cols



from sklearn.preprocessing import PolynomialFeatures
education_levels = ['tertiary', 'secondary', 'primary', 'unknown']
ordinal_transformer = OrdinalEncoder(categories=[education_levels], dtype=int)

# numeric_transformer = make_pipeline(PolynomialFeatures(degree=1),StandardScaler())
numeric_transformer = StandardScaler()

binary_transformer = make_pipeline(OneHotEncoder(dtype=int, drop='if_binary'))
```

```
categorical_transformer = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
```

```
import seaborn as sns
cor = X_train_trans.corr()
sns.heatmap(cor)

cor.style.highlight_between(left=-0.9999, right=-0.7)
```

```
HGB_preprocessor = make_column_transformer(
        (numeric_transformer, num_cols),
        (ordinal_transformer, ord_cols),
        # ('binary', binary_transformer, binary_features),
        (categorical_transformer, cat_cols),
        # ('drop', 'passthrough', drop_features)
    remainder='passthrough'
    )


baseline = make_pipeline(
    preprocessor,
    DummyClassifier()
)

baseline_result = pd.DataFrame(
    cross_validate(
        baseline,
        X_train,
        y_train,
        cv=10,
        scoring='roc_auc',
        return_train_score=True
    )
)
baseline_result.mean()
```

```
HGB = HistGradientBoostingClassifier(random_state=RANDOM_STATE
                                     , categorical_features=ord_cols+cat_cols)


HGB_result = pd.DataFrame(
    cross_validate(
        HGB,
        X_train,
        y_train,
        cv=10,
        scoring='roc_auc',
        return_train_score=True
    )
)
HGB_result.mean()
```

```
RF = make_pipeline(
    preprocessor,
    RandomForestClassifier(random_state=RANDOM_STATE
```

```
                           ,max_depth=5)
)

RF_result = pd.DataFrame(
    cross_validate(
        RF,
        X_train,
        y_train,
        cv=10,
        scoring='roc_auc',
        return_train_score=True
    )
)
RF_result.mean()
```

```
param_dist = {
    "randomforestclassifier__n_estimators": [50, 100, 200],
    "randomforestclassifier__max_depth":[3, 5, 7, 10],
    "randomforestclassifier__max_features": ["sqrt", "log2"]
}


RF_grid_search = GridSearchCV(RF,
                              param_dist,
                              n_jobs=-1,
                              cv=5,
                              scoring='roc_auc',
                              return_train_score=True
                             )

RF_grid_search.fit(X_train, y_train)

results_RF = pd.DataFrame({
    'mean_test_score': RF_grid_search.cv_results_['mean_test_score'],
    'std_test_score': RF_grid_search.cv_results_['std_test_score'],
    'mean_train_score': RF_grid_search.cv_results_['mean_train_score'],
    'std_train_score': RF_grid_search.cv_results_['std_train_score'],
    'params': RF_grid_search.cv_results_['params']}
)
RF_grid_search.best_index_
RF_grid_search.best_params_
```

```
param_dist = {
    # "l2_regularization": loguniform(1e-5, 1e2),
    "max_iter": randint(30, 200),
    "max_depth": randint(3, 20),

}

HGB_random_search = RandomizedSearchCV(HGB,
                                       param_dist,
                                       n_iter=100,
                                       n_jobs=-1,
                                       cv=5,
                                       scoring='roc_auc',
                                       return_train_score=True,
                                       random_state=RANDOM_STATE
                                      )

HGB_random_search.fit(X_train, y_train)

results_HGB = pd.DataFrame({
```

```
    'mean_test_score': HGB_random_search.cv_results_['mean_test_score'],
    'std_test_score': HGB_random_search.cv_results_['std_test_score'],
    'mean_train_score': HGB_random_search.cv_results_['mean_train_score'],
    'std_train_score': HGB_random_search.cv_results_['std_train_score'],
    'params': HGB_random_search.cv_results_['params']}
)
results_HGB.sort_values(by='mean_test_score', ascending=False)

prediction = HGB_random_search.predict(X_train)
from sklearn.metrics import accuracy_score
accuracy_score(y_train, prediction)
```

```
from lightgbm import LGBMClassifier

# Assuming num_cols, ord_cols, and cat_cols are already defined
numeric_transformer = StandardScaler()
ordinal_transformer = OrdinalEncoder()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = make_column_transformer(
    (numeric_transformer, num_cols),
    (ordinal_transformer, ord_cols),
    (categorical_transformer, cat_cols),
    remainder='passthrough'
)

# Define the LightGBM model pipeline
LGBM = make_pipeline(
    preprocessor,
    LGBMClassifier(random_state=RANDOM_STATE, verbose=-1)
)

# Perform cross-validation
LGBM_result = pd.DataFrame(
    cross_validate(
        LGBM,
        X_train,
        y_train,
        cv=10,
        scoring='roc_auc',
        return_train_score=True
    )
)
print(LGBM_result.mean())

# Parameter grid for GridSearchCV
param_dist = {
    "lgbmclassifier__n_estimators": [50, 100, 200],
    "lgbmclassifier__max_depth": [3, 5, 7, 10],
    "lgbmclassifier__num_leaves": [31, 62, 127]
}

# Grid search setup
LGBM_grid_search = GridSearchCV(LGBM,
                                param_dist,
                                n_jobs=-1,
                                cv=5,
                                scoring='roc_auc',
                                return_train_score=True
                               )

# Fit the grid search
LGBM_grid_search.fit(X_train, y_train)
print(LGBM_grid_search.best_score_)
print(LGBM_grid_search.best_params_)


# Collect results
```

```
results_LGBM = pd.DataFrame({
    'mean_test_score': LGBM_grid_search.cv_results_['mean_test_score'],
    'std_test_score': LGBM_grid_search.cv_results_['std_test_score'],
    'mean_train_score': LGBM_grid_search.cv_results_['mean_train_score'],
    'std_train_score': LGBM_grid_search.cv_results_['std_train_score'],
    'params': LGBM_grid_search.cv_results_['params']
})
```

```
print(f"HGB accuracy:{accuracy_score(y_test, HGB_random_search.predict(X_test)):.4f}")
print(classification_report(y_test,  HGB_random_search.predict(X_test), digits=4))

ConfusionMatrixDisplay.from_estimator(HGB_random_search, X_test, y_test)
```

```
model_dict = {'RF_grid_search': RF_grid_search,
              'HGB_random_search': HGB_random_search,
              'LGBM_grid_search': LGBM_grid_search}
compute_and_plot_roc_curve(X_test, y_test, figsize=(10,10), **model_dict)
```

```
from lightgbm import plot_importance
X_train_trans = preprocessor.fit_transform(X_train)
column_names = (
    preprocessor.named_transformers_['standardscaler'].get_feature_names_out().tolist() +
    preprocessor.named_transformers_['ordinalencoder'].get_feature_names_out().tolist() +
    preprocessor.named_transformers_['onehotencoder'].get_feature_names_out().tolist()
    )
X_train_trans = pd.DataFrame(X_train_trans, columns=column_names)
lgbm = LGBMClassifier(random_state=RANDOM_STATE
                      , verbose=-1
                      , max_depth=3
                      , n_estimators=200
                      , num_leaves=31)
lgbm.fit(X_train_trans, y_train)
plot_importance(lgbm)
```

```
pd.DataFrame({'features':column_names,

 'importance':RF_grid_search.best_estimator_.named_steps['randomforestclassifier'].feature_importances_}).
sort_values(by='importance').plot.bar(x='features')
```

```
plot_permutation_importance(RF_grid_search.best_estimator_, X_train, y_train)
```

```
import shap
explainer = shap.Explainer(RF_grid_search.best_estimator_[-1])
shap_values = explainer.shap_values(X_train_trans)
shap_values
```

```
from sklearn.inspection import permutation_importance

def plot_permutation_importance(clf, X, y):
    fig, ax = plt.subplots(figsize=(7, 6))
    result = permutation_importance(clf, X, y, n_repeats=10, random_state=RANDOM_STATE, n_jobs=-1)
    perm_sorted_idx = result.importances_mean.argsort()

    ax.boxplot(
```

```
        result.importances[perm_sorted_idx].T,
        vert=False,
        labels=X.columns[perm_sorted_idx],
    )
    ax.axvline(x=0, color="k", linestyle="--")
    ax.figure.tight_layout()
    plt.show()
    # return ax
```