Demonstrate Pre processing (Data Cleaning, Integration and Transformation) activity on suitable data:

- Identify and Delete Rows that Contain Duplicate Data by considering an appropriate dataset.
- Identify and Delete Columns That Contain a Single Value by considering an appropriate dataset.

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OrdinalEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Load the housing dataset
housing = pd.read_csv('housing.csv')

# Data Cleaning - Identifying and Deleting Duplicate Rows
housing_cleaned = housing.drop_duplicates()
print("Number of rows after removing duplicates:", len(housing_cleaned))

# Data Integration - Identifying and Deleting Columns with a Single Value
columns_to_drop = []
for column in housing_cleaned.columns:
    if housing_cleaned[column].nunique() == 1:
        columns_to_drop.append(column)
housing_integrated = housing_cleaned.drop(columns=columns_to_drop)
print("Columns after removing single-valued columns:")
print(housing_integrated.columns)

# Data Transformation - Preprocessing Pipeline
num_attribs = list(housing_integrated.select_dtypes(include=['float64',
'int64']))
cat_attribs = list(housing_integrated.select_dtypes(include=['object']))

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
    ('ordinal_encoder', OrdinalEncoder()),
    ('one_hot_encoder', OneHotEncoder()),
])

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", cat_pipeline, cat_attribs),
])

housing_preprocessed = full_pipeline.fit_transform(housing_integrated)
print("Preprocessed data shape:", housing_preprocessed.shape)
```

Demonstrate the working of Ensembling classifier for a suitable data set

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

X, y = make_moons(n_samples=100, noise=0.15)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('rf', rnd_clf),
('svm', svm_clf)], voting='hard')
voting_clf.fit(X_train, y_train)

y_pred_voting = voting_clf.predict(X_test)
print("Voting Classifier Accuracy:", accuracy_score(y_test, y_pred_voting))
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris['data'],
iris['target'], test_size=0.3, random_state=42)

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('rf', rnd_clf),
('svm', svm_clf)], voting='hard')
voting_clf.fit(X_train, y_train)

y_pred_voting = voting_clf.predict(X_test)
print("Voting Classifier Accuracy:", accuracy_score(y_test, y_pred_voting))
```