Open the dataset `ames_housing_no_missing.csv` with the following command:

```
import pandas as pd

ames_housing = pd.read_csv("../datasets/ames_housing_no_missing.csv")



target_name = "SalePrice"

data, target = ames_housing.drop(columns=target_name),
ames_housing[target_name]

target = (target > 200_000).astype(int)
```

`ames_housing` is a pandas dataframe. The column "SalePrice" contains the target variable.

We did not encounter any regression problem yet. Therefore, we convert the regression target into a classification target to predict whether or not an house is expensive. "Expensive" is defined as a sale price greater than $200,000.

Question 1

(1/1 point)
Use the `data.info()` and `data.head()` commands to examine the columns of the dataframe. The dataset contains:
a) only numerical features b) only categorical features c) both numerical and categorical features c) both numerical and categorical features - correct

EXPLANATION

Solution: c)

To answer to the question, you can check the dataframe information with the following command:

```
data.info()
```

and

```
data.head()
```

The "Dtype" column gives information of the data type. We observe three different data types: "object", "int64", and "float64".

"float64" columns typically represent numerical data. "object" columns usually store string labels for categorical values. "int64" can either represent numerical quantitities or categorical codes. The actual meaning depends and one would often need to look at the actual values of the first lines or at the dataset documentation to make sure whether this column should be treated as a numerical or categorical quantity.

Therefore this dataset has both numerical (e.g. "LotFrontage") and categorical (e.g. "BldgType") features.

Hide Answer

You have used 1 of 1 submissions

Question 2

(1 point possible)

How many features are available to predict whether or not a house is expensive ?

a) 79 b) 80 b) 80 - incorrect c) 81

EXPLANATION

Solution: a)

From the original dataframe, we should exclude the column "SalePrice" because it corresponds to the target that we want to predict.

The resulting dataframe named "data" has 79 columns as seen with the `data.shape` attribute or by measuring `len(data.columns)` for instance.

Hide Answer

You have used 1 of 1 submissions

Question 3

(1/1 point)

How many features are represented with numbers?

a) 0 b) 36 b) 36 - correct c) 42 d) 79

Hint: You can use the method `df.select_dtypes` or the function `sklearn.compose.make_column_selector` as shown in the notebook.

EXPLANATION

Solution: b)

You can only select columns corresponding to data type `int64` and `float64`. The following code will filter these columns:

```
data_numbers = data.select_dtypes(["integer", "float"])
data_numbers.info()
```

In this case, 36 columns will be available. Programmatically, we can also access the number of columns with:

```
len(data_numbers.columns)
```

Hide Answer

You have used 1 of 1 submissions

Question 4

(1 point possible)

Refer to the dataset description regarding the meaning of the dataset.

Among the following columns, which columns express a quantitative numerical value (excluding ordinal categories)?

a) “LotFrontage” b) “LotArea” c) “OverallQual” d) “OverallCond” e) “YearBuilt”
c) “OverallQual”, d) “OverallCond”, e) “YearBuilt”, - incorrect

Select all answers that apply

EXPLANATION

Solution: a) b) e)

"OverallQual" and "OverallCond" are ordinal categorical variables. While technically "YearBuilt" is more a date than a quantity, it is fine for machine learning models to treat it as a quantity because the year of construction is directly related to the age of the house.

Hide Answer

You have used 2 of 2 submissions

We consider the following numerical columns:

```
numerical_features = [
    "LotFrontage", "LotArea", "MasVnrArea", "BsmtFinSF1", "BsmtFinSF2",
    "BsmtUnfSF", "TotalBsmtSF", "1stFlrSF", "2ndFlrSF", "LowQualFinSF",
```

```
    "GrLivArea", "BedroomAbvGr", "KitchenAbvGr", "TotRmsAbvGrd", "Fireplaces",
    "GarageCars", "GarageArea", "WoodDeckSF", "OpenPorchSF", "EnclosedPorch",
    "3SsnPorch", "ScreenPorch", "PoolArea", "MiscVal",
]
```

Question 5

(1/1 point)
Now create a predictive model that uses these numerical columns as input data. Your predictive model should be a pipeline composed of a `sklearn.preprocessing.StandardScaler` to scale these numerical data and a `sklearn.linear_model.LogisticRegression`.
What is the accuracy score obtained by 10-fold cross-validation (you can set the parameter `cv=10` when calling `cross_validate`) of this pipeline?
a) ~0.5 b) ~0.7 c) ~0.9 c) ~0.9 - correct

EXPLANATION

Solution: c)

The code to get the score is the following:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_validate

data_numerical = data[numerical_features]

model = make_pipeline(StandardScaler(), LogisticRegression())
```

```
cv_results_num = cross_validate(model, data_numerical, target,
cv=10)
test_score_num = cv_results_num["test_score"]
test_score_num.mean()
```

Hide Answer

You have used 1 of 1 submissions

Question 6

(1 point possible)

Instead of solely using the numerical columns, let us build a pipeline that can process both the numerical and categorical features together as follows:

- the `numerical_features` (as defined above) should be processed as previously done with a `StandardScaler`;
- the left-out columns should be treated as categorical variables using a `sklearn.preprocessing.OneHotEncoder`. To avoid any issue with rare categories that could only be present during the prediction, you can pass the parameter `handle_unknown="ignore"` to the `OneHotEncoder`.

What is the accuracy score obtained by 10-fold cross-validation of the pipeline using both the numerical and categorical features?

a) ~0.7 b) ~0.9 c) ~1.0c) ~1.0 - incorrect

EXPLANATION

Solution: b)

The code to get the score is the following:

```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder

categorical_features = data.columns.difference(numerical_features)
```

```
categorical_processor = OneHotEncoder(handle_unknown="ignore")
numerical_processor = StandardScaler()

preprocessor = make_column_transformer(
    (categorical_processor, categorical_features),
    (numerical_processor, numerical_features),
)
model = make_pipeline(preprocessor,
LogisticRegression(max_iter=1_000))
cv_results_all = cross_validate(model, data, target, cv=10)
test_score_all = cv_results_all["test_score"]
test_score_all.mean()
```

From this analysis, we observe that the mean test score of the model taking into account both numerical and categorical features is higher than only using the numerical features.

Hide Answer

You have used 1 of 1 submissions

Question 7

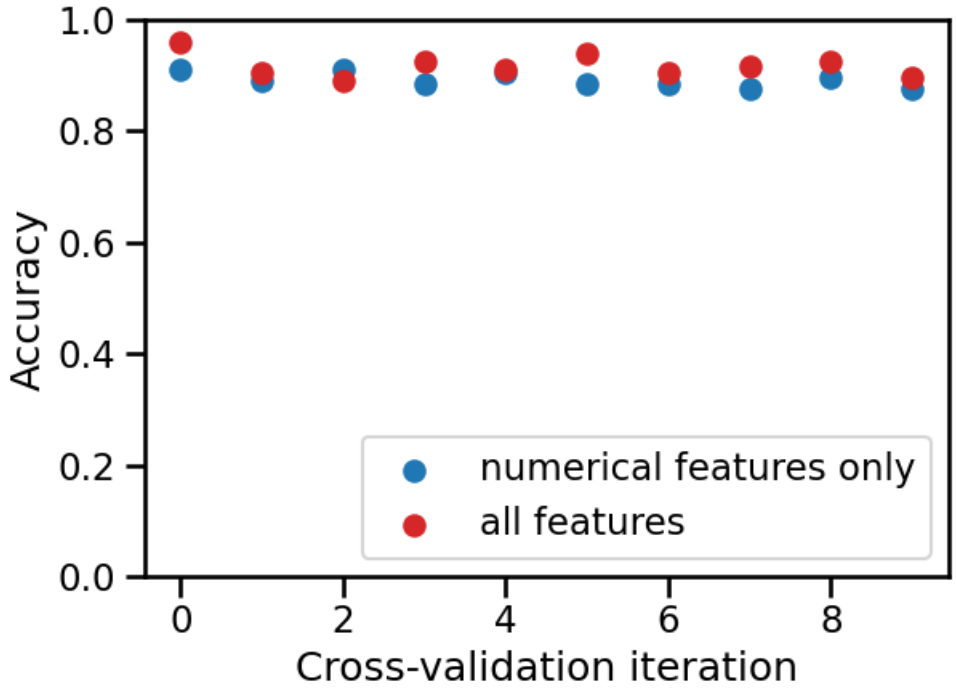
(1/1 point)

One way to compare two models is by comparing their means, but small differences in performance measures might easily turn out to be merely by chance (e.g. when using random resampling during cross-validation), and not because one model predicts systematically better than the other.

Another way is to compare cross-validation test scores of both models fold-to-fold, i.e. counting the number of folds where one model has a better test score

than the other. This provides some extra information: are some partitions of the data making the classifaction task particularly easy or hard for both models?

Let's visualize the second approach.



Select the true statement.

The number of folds where the model using all features perform better than the model using only numerical features lies in the range:

a) [0, 3]: the model using all features is consistently worse b) [4, 6]: both models are almost equivalent c) [7, 10]: the model using all features is consistently better c) [7, 10]: the model using all features is consistently better - correct

EXPLANATION

solution: c)

To answer the question, we can now compare the score of each fold to investigate if this improvement is generally happening on all folds of the cross-validation:

```
import numpy as np
import matplotlib.pyplot as plt

indices = np.arange(len(test_score_num))
plt.scatter(
    indices, test_score_num, color="tab:blue", label="numerical
features only"
)
plt.scatter(
    indices,
    test_score_all,
    color="tab:red",
    label="all features",
)
plt.ylim((0, 1))
plt.xlabel("Cross-validation iteration")
plt.ylabel("Accuracy")
_ = plt.legend(loc="lower right")

print(
    "A model using all features is better than a"
    " model using only numerical features for"
    f" {sum(test_score_all > test_score_num)} CV iterations out of 10."
)
```

We observe that 9 times out of 10, the model based on both numerical and categorical features is better than the model that only uses numerical features.