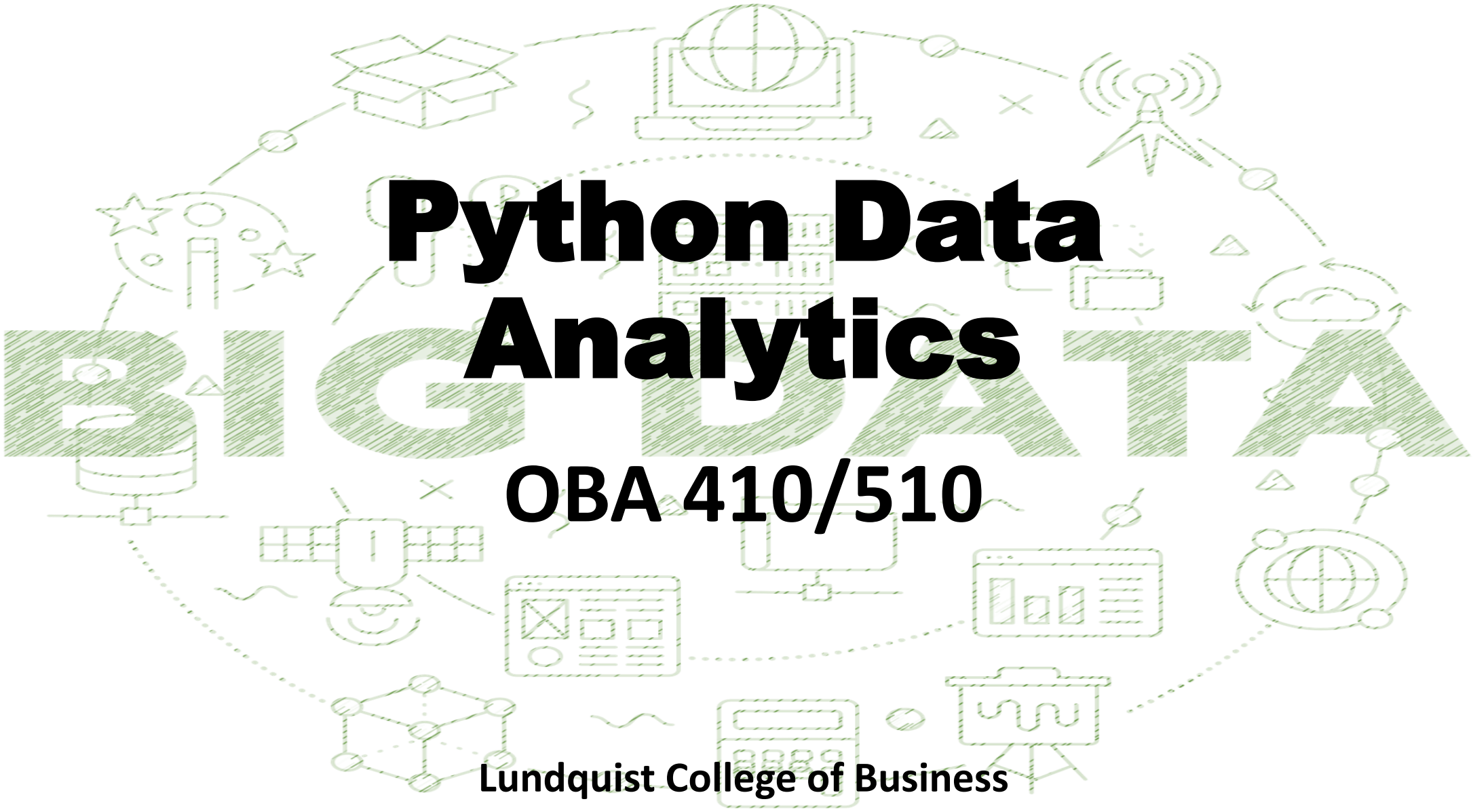

Python Data Analytics

OBA 410/510

Lundquist College of Business

Background image from www. givweb.com

Some of the slides and contents are adopted from Statistical Learning course by Dr. Trevor Hastie

UNIVERSITY OF OREGON
Lundquist College of Business

Handling Categorical Features

Handling Categorical Features

- So far, all features in our datasets have been numeric, because models in scikit-learn only take numeric features
- For many applications, we may have categorical features (discrete features)
- Examples of categorical features: brand of a product, education level, gender, …
- Sample data (adult incomes in the US):

	age	workclass	education	gender	hours-per-week	occupation	income
0	39	State-gov	Bachelors	Male	40	Adm-clerical	<=50K
1	50	Self-emp-not-inc	Bachelors	Male	13	Exec-managerial	<=50K
2	38	Private	HS-grad	Male	40	Handlers-cleaners	<=50K
3	53	Private	11th	Male	40	Handlers-cleaners	<=50K

One-Hot-Encoding

- By far the most common way to represent categorical variables is using the ***one-hot-encoding*** or ***one-out-of-V encoding***, also known as ***dummy variables***
- A categorical variable is replaced with one or more new features that can have values 0 and 1
- We can represent any number of categories (in a categorical variable) by introducing one new feature per category
- Let's say for the workclass feature in our sample data we have possible values of "Government Employee", "Private Employee", and "Self Employed"

One-Hot-Encoding

- To encode these three possible values, we create three new features called, “Government Employee”, “Private Employee”, and “Self Employed”
- A feature is 1 if workclass for this person has the corresponding value and 0 otherwise
- So, exactly one of these three features will be 1 for each data point
- This is why this method is called one-hot or one-out-of-N encoding
- When we use this data to develop our models, we **would drop the original workclass feature and only keep the dummy(0-1)** features

One-Hot-Encoding

- To encode these three possible values, we create three new features called, “Government Employee”, “Private Employee”, and “Self Employed”

Table 4-2. Encoding the workclass feature using one-hot encoding

workclass	Government Employee	Private Employee	Self Employed
Government Employee	1	0	0
Private Employee	0	1	0
Self Employed	0	0	1

- We can use pandas to encode our categorical features
- ***get_dummies*** function in pandas automatically transforms all categorical features
- Before encoding, it is always a good idea to check the categories of categorical features

One-Hot-Encoding

- Be carful to **separate target variable before encoding** your features
- Make sure to **encode the whole data before splitting to train and test sets**

```
1  income=pd.read_csv('Adults_Income.csv')
2  income.head()
```

	age	workclass	occupation	hours-per-week	income
0	20	Private	Sales	44	<=50K
1	31	Private	Sales	38	>50K
2	24	Private	Tech-support	50	<=50K
3	43	Private	Tech-support	40	>50K
4	30	Private	Sales	40	<=50K

One-Hot-Encoding

- .

```
1 income['workclass'].value_counts()
```

```
Private         1986
Self-emp-inc     180
Federal-gov       51
Name: workclass, dtype: int64
```

```
1 income['occupation'].value_counts()
```

```
Sales           1742
Tech-support     475
Name: occupation, dtype: int64
```

One-Hot-Encoding

- .

```
1  X, y=income.iloc[:,:-1],income.iloc[:,-1]
```

```
1  X_dummies=pd.get_dummies(X)
2  X_dummies.head()
```

	age	hours-per-week	workclass_ Federal-gov	workclass_ Private	workclass_ Self-emp-inc	occupation_ Sales	occupation_ Tech-support
0	20	44	0	1	0	1	0
1	31	38	0	1	0	1	0
2	24	50	0	1	0	0	1
3	43	40	0	1	0	0	1
4	30	40	0	1	0	1	0

One-Hot-Encoding

In [9]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

In [10]:
```
X_train, X_test, y_train, y_test=train_test_split(X_income_dummies,y_income,random_state=0)
```

In [19]:
```
knn=KNeighborsClassifier(n_neighbors=11)
knn.fit(X_train, y_train)
```

Out[19]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=11, p=2,
           weights='uniform')
```

In [20]:
```
print('knn on train {:.2%}'.format(knn.score(X_train,y_train)))
print('knn on test {:.2%}'.format(knn.score(X_test,y_test)))
```

```
knn on train 76.71%
knn on test 73.15%
```

One-Hot-Encoding

- Making predictions:
  - We have to pass the data in the same structure that we used to train our model, and that is X_dummies structure

```
# Predictions
# intial features structure:
# age workclass occupation hours-per-week
# p1=[25,'private','Tech-support',30]
```

```
# X_dummies structure
# age, hours-per-week, workclass_ Federal-gov, workclass_ Private ,workclass_ Self-emp-inc ,occupation_ Sales, occupation_ Tech-s
p1=[25,30,0,1,0,0,1]
```

```
knn.predict([p1])
```

```
array([' <=50K'], dtype=object)
```

Numbers Can Encode Categoricals

- Sometimes you might have integer values for a feature, while that feature is actually a discrete variable
- For instance data about a product type might have been collated like:
  - book: 1
  - CD: 2
  - Pen: 3
- In these situations, we have to first change the data type to string, then use ***get_dummies*** function to encode the feature, otherwise python would treat that feature as a numeric feature

Numbers Can Encode Categoricals

```
# create a DataFrame with an integer feature and a categorical string feature
demo_df = pd.DataFrame({'Prodcut Type': [3, 1, 2, 1],
                        'Prodcut Colour': ['blue', 'red', 'red', 'blue']})
demo_df
```

	Prodcut Type	Prodcut Colour
0	3	blue
1	1	red
2	2	red
3	1	blue

Numbers Can Encode Categoricals

- .

```
pd.get_dummies(demo_df)
```

	Prodcut Type	Prodcut Colour_blue	Prodcut Colour_red
0	3	1	0
1	1	0	1
2	2	0	1
3	1	1	0

Numbers Can Encode Categoricals

- ```
  demo_df['Prodcut Type']=demo_df['Prodcut Type'].astype(str)
  ```

  ```
  pd.get_dummies(demo_df)
  ```

	Prodcut Type_1	Prodcut Type_2	Prodcut Type_3	Prodcut Colour_blue	Prodcut Colour_red
0	0	0	1	1	0
1	1	0	0	0	1
2	0	1	0	0	1
3	1	0	0	1	0
  ```

One-Hot-Encoding (WestRoxbury_categorical data)


```
house=pd.read_csv('WestRoxbury_categorical.csv')
house.head()
```

	TOTAL VALUE	LOT SQFT	YR BUILT	GROSS AREA	LIVING AREA	FLOORS	ROOMS	BEDROOMS	FULL BATH	HALF BATH	KITCHEN	FIREPLACE	REMODEL
0	344.2	9965	1880	2436	1352	2.0	6	3	1	1	1	No	No
1	412.6	6590	1945	3108	1976	2.0	10	4	2	1	1	No	Yes_Recently
2	330.1	7500	1890	2294	1371	2.0	8	4	1	1	1	No	No
3	498.6	13773	1957	5032	2608	1.0	9	5	1	1	1	Yes	No
4	331.5	5000	1910	2370	1438	2.0	7	3	2	0	1	No	No

One-Hot-Encoding (WestRoxbury_categorical data)

UNIVERSITY OF OREGON
Lundquist College of Business

```
X,y=house.iloc[:,1:],house['TOTAL VALUE ']
```

```
X_dummies=pd.get_dummies(X)
```

```
X_dummies.head()
```

	LOT SQFT	YR BUILT	GROSS AREA	LIVING AREA	FLOORS	ROOMS	BEDROOMS	FULL BATH	HALF BATH	KITCHEN	FIREPLACE_No	FIREPLACE_Yes	REMODEL_No	REMODEL_Yes
0	9965	1880	2436	1352	2.0	6	3	1	1	1	1	0	1	0
1	6590	1945	3108	1976	2.0	10	4	2	1	1	1	0	0	0
2	7500	1890	2294	1371	2.0	8	4	1	1	1	1	0	1	0
3	13773	1957	5032	2608	1.0	9	5	1	1	1	0	1	1	0
4	5000	1910	2370	1438	2.0	7	3	2	0	1	1	0	1	0

One-Hot-Encoding (WestRoxbury_categorical data)

UNIVERSITY OF OREGON
Lundquist College of Business

•

```
X_train, X_test, y_train, y_test=train_test_split(X_dummies,y,random_state=0)
```

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,y_train)
print('reg acc on train: {:.3f}'.format(lr.score(X_train,y_train)))
print('reg acc on test: {:.3f}'.format(lr.score(X_test,y_test)))
```

```
reg acc on train: 0.818
reg acc on test: 0.802
```

One-Hot-Encoding (WestRoxbury_categorical data)

UNIVERSITY OF OREGON
Lundquist College of Business

- .

```
smpl1=X_test.iloc[:4]
smpl1
```

	LOT SQFT	YR BUILT	GROSS AREA	LIVING AREA	FLOORS	ROOMS	BEDROOMS	FULL BATH	HALF BATH	KITCHEN	FIREPLACE_No	FIREPLACE_Yes	REMODEL_No	REMODEL_Yes
1519	4026	1940	2520	1047	1.0	6	3	1	1	1	0	1	1	0
3457	7000	1848	6235	3446	2.0	12	5	3	0	1	0	1	1	0
895	4615	1956	2304	1306	1.5	6	3	1	1	1	0	1	1	0
5423	5000	1930	1930	1273	2.0	7	3	1	0	1	0	1	1	0

```
lr.predict(smpl1)
```

```
array([304.37214016, 639.19502363, 337.12378227, 327.2775603 ])
```