Pitfalls To Avoid while Interpreting Machine Learning-PDP/ICE case

Satya Pattnaik Oct 22, 2020 · 9 min read

Photo by Leio McLaren (@leiomclaren) on Unsplash

Abstract

One can find numerous articles today on Explainable AI, some of which can be found **here.** The most standard guide for Explainable AI will undoubtedly be **this book** by **Christoph Molnar.** When I came across the recent paper **Pitfalls to Avoid when**

This article is focused on the pitfalls we need to avoid while interpreting Partial Dependence Plots(PDPs)/Individual Conditional Expectation(ICE) plots. These are post hoc techniques used to observe how the model takes a decision by keeping all exogenous variables fixed, except one(also two in case of PDPs) which is regarded as the **feature of interest**. This variable is allowed to take all the possible values and we observe its marginal effect on the model's decision making. To have a proper understanding please refer to **this**.

Definitions

Feature Dependence is an issue while we use Additive models, as multicollinearity can cause an ordinary least square to fail or make it difficult for the Data scientist to interpret the coefficients of the model. But this is not the case with tree-based models, whether it is a Decision Tree or Bagged Ensembles(e.g. Random Forest) or Boosted Trees(like a Gradient Boosting Machine). We never care to go for a stern check of Feature Dependence between the exogenous variables while using such models. Why should we? Trees work based on Univariate Feature Split and eventually multicollinearity does not create havoc as in the case of a Logit.

However, the working of PDP/ICE assumes that the feature of interest(s) and the complementary set of features should be independent. Why should they be independent? PDP/ICE techniques are based on perturbation of data points. The perturbation of **feature of interest** is made by:

- Replacement from equidistant Grid of that feature.
- Random Sampling of Values.
- Replacement using quantiles. (Our plots will be based on Percentiles)

In all three cases, dependent features when perturbed can result in a joint distribution that will have extrapolated data points that are infeasible in the real world. Explaining the marginal contribution of a feature in such a case can lead to spurious interpretations. We will observe this phenomenon very soon in our example.

Data and Problem Statement

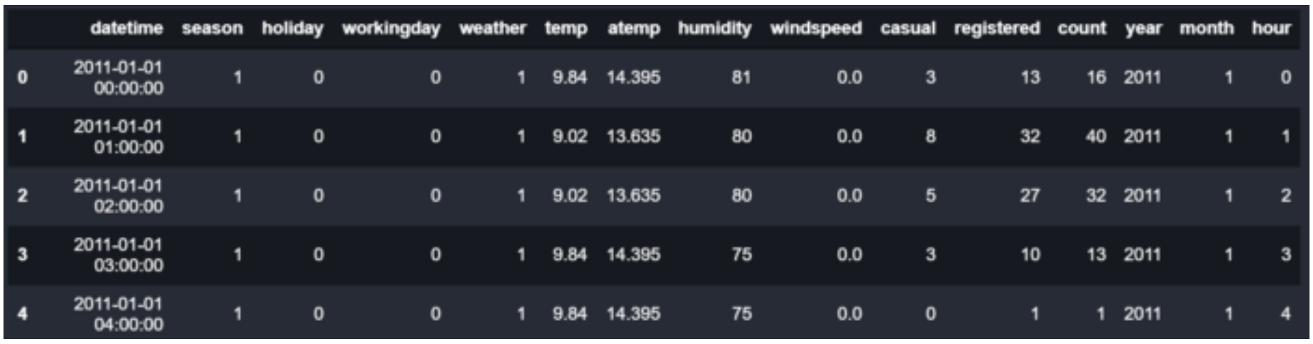
We begin our analysis with the **bike-sharing dataset** from **Kaggle.** The location from which the bike-sharing information is recorded in Washington D.C., in **Northern** corresponding **notebook** for this article.

We begin by importing libraries

```
import numpy as np
import pandas as pd
import calendar
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from yellowbrick.target import FeatureCorrelation
from yellowbrick.regressor import ResidualsPlot
from pdpbox import pdp, info_plots
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 100
```

We load the dataset, extract the time-based features, and review it.

```
df = pd.read_csv('train.csv')
df['datetime'] = pd.to_datetime(df['datetime'])#Convert to Pandas
Datetime Type
df['year'] = df['datetime'].dt.year#Extract the Year as a Feature
df['month'] = df['datetime'].dt.month#Extract the Month as a Feature
df['hour'] = df['datetime'].dt.hour#Extract the Hour as a Feature
df.head() #Print the top 5 rows of the dataframe
```



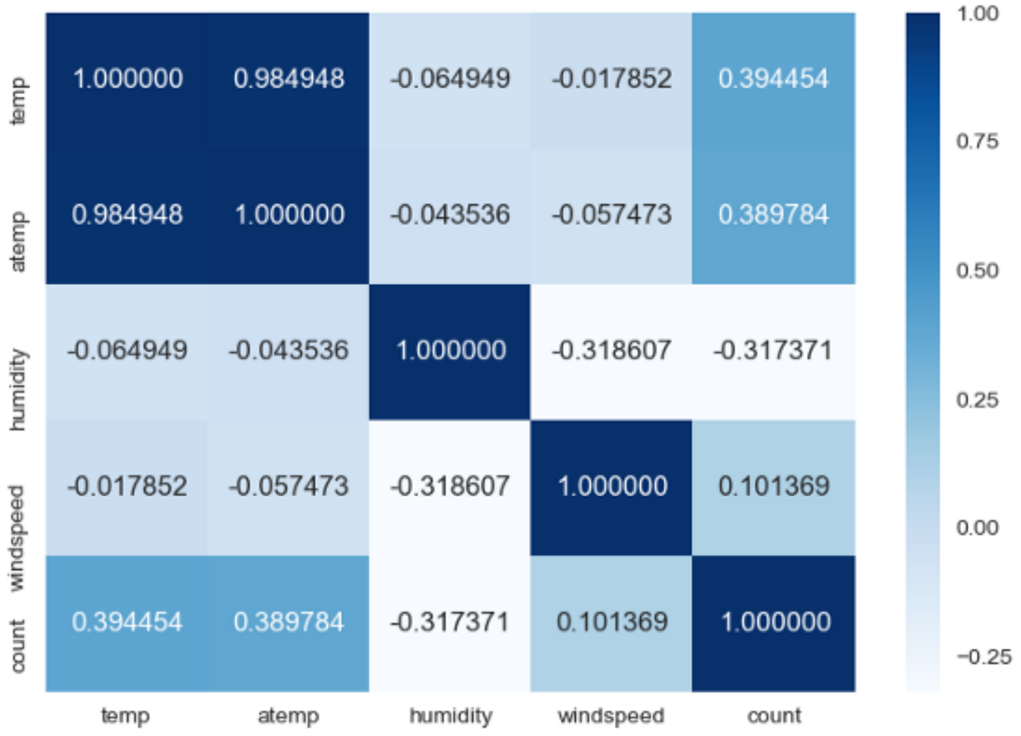
	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	year	month	hour
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	2011	1	0
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	2011	1	1
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	2011	1	2
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13	2011	1	3
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1	2011	1	4

A sample of Data(Source: Author)

Feature Dependence(Continuous Variables)

We compute the Pearson Correlation for continuous variables. We observe that the temperature('temp') and "feel like" temperature('atemp') are highly correlated. This

```
#Pearson Correlation for Continiuos Variables
continious_variables = df[['temp', 'atemp', 'humidity',
'windspeed','count']]
corr = continious_variables.corr()
corr = sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns,
           annot=True, fmt="f",cmap="Blues")
```



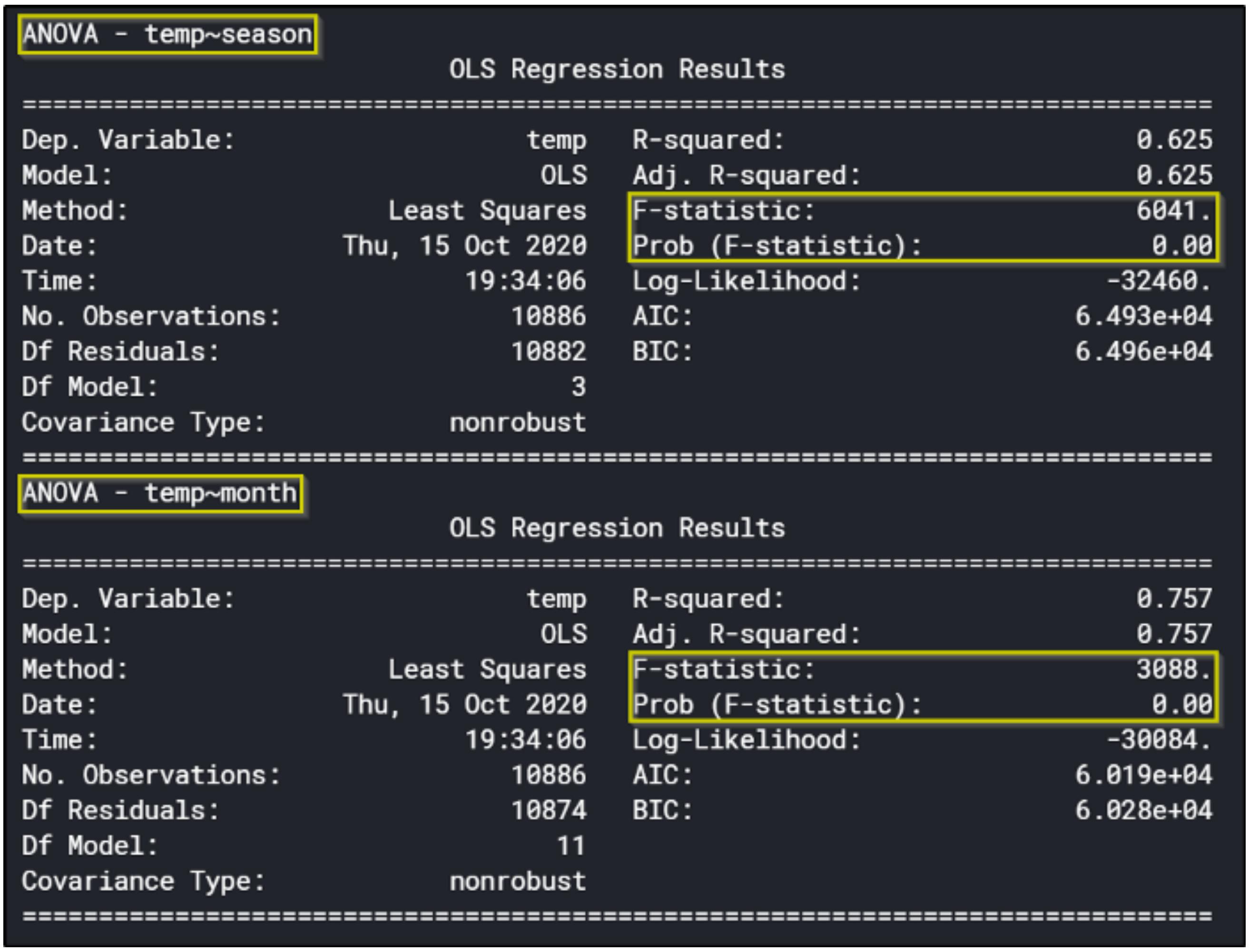
	temp	atemp	humidity	windspeed	count
temp	1.000000	0.984948	-0.064949	-0.017852	0.394454
atemp	0.984948	1.000000	-0.043536	-0.057473	0.389784
humidity	-0.064949	-0.043536	1.000000	-0.318607	-0.317371
windspeed	-0.017852	-0.057473	-0.318607	1.000000	0.101369
count	0.394454	0.389784	-0.317371	0.101369	1.000000

Pearson Correlation(Source: Author)

Feature Dependence(Categorical Variable to Continuous Variable)

We observe the dependency between a continuous feature — the temperature factor('temp') and two categorical factors — the season('season') and month('month') of the rental booking. The ANOVA table for both of the dependencies i.e. 'temp' with 'season' and 'temp' with 'month' is computed. We observe that the P-values for the F-Statistic in both of the cases is zero. This suggests that group means of temperature('temp') are statistically different across the categories of 'month' and 'season'. This suggests that the features are **linearly dependent**. This is pretty intuitive as the temperature means(average) across months/seasons will certainly vary.

```
#Fit an OLS with temperature as the continious target variable
#and season as the explanatory categorical variable.
model = smf.ols(formula='temp ~ C(season)', data=df)
res = model.fit()
print('ANOVA - temp~season')
summary = res.summary() #OLS Summary
print(summary.tables[0]) #Print the Anova Table only

#Fit an OLS with temperature as the continious target variable
#and month as the explanatory categorical variable.
model = smf.ols(formula='temp ~ C(month)', data=df)
res = model.fit()
print('ANOVA - temp~month')
summary = res.summary() #OLS SUmmary
print(summary.tables[0]) #Print the Anova Table only
```



```
ANOVA - temp~season
                            OLS Regression Results
==============================================================================
Dep. Variable:                   temp   R-squared:                       0.625
Model:                            OLS   Adj. R-squared:                  0.625
Method:                 Least Squares   F-statistic:                     6041.
Date:                Thu, 15 Oct 2020   Prob (F-statistic):               0.00
Time:                        19:34:06   Log-Likelihood:                -32460.
No. Observations:               10886   AIC:                         6.493e+04
Df Residuals:                   10882   BIC:                         6.496e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
ANOVA - temp~month
                            OLS Regression Results
==============================================================================
Dep. Variable:                   temp   R-squared:                       0.757
Model:                            OLS   Adj. R-squared:                  0.757
Method:                 Least Squares   F-statistic:                     3088.
Date:                Thu, 15 Oct 2020   Prob (F-statistic):               0.00
Time:                        19:34:06   Log-Likelihood:                -30084.
No. Observations:               10886   AIC:                         6.019e+04
Df Residuals:                   10874   BIC:                         6.028e+04
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
```

ANOVA Table(Source: Author)

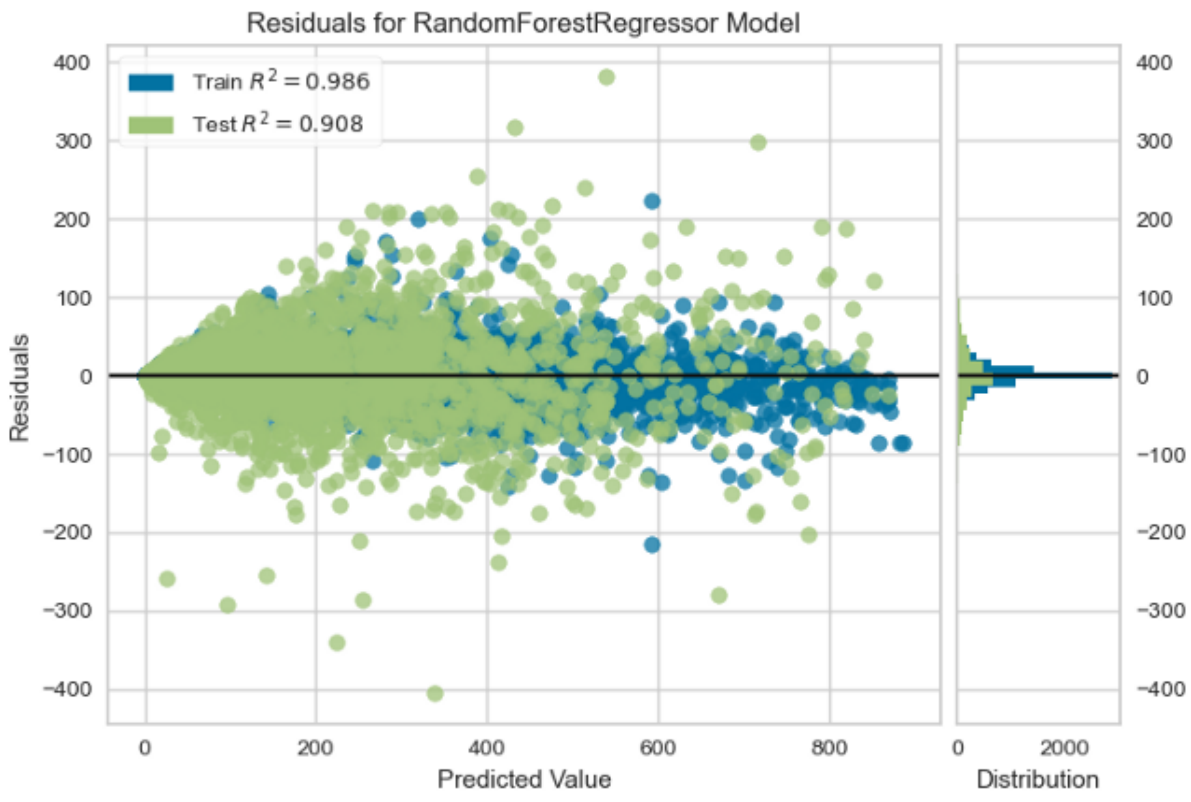
Partial Dependence Plot(no correlated variables)

We begin by fitting a Random Forest Regressor, making sure that we don't take any continuous or categorical dependent variables. We make sure that we have a good fit.

values of both the train and test. Both of these factors suggest a good fit. A well-fitted model is absolutely necessary in order to explain the predictive decisions of the model.

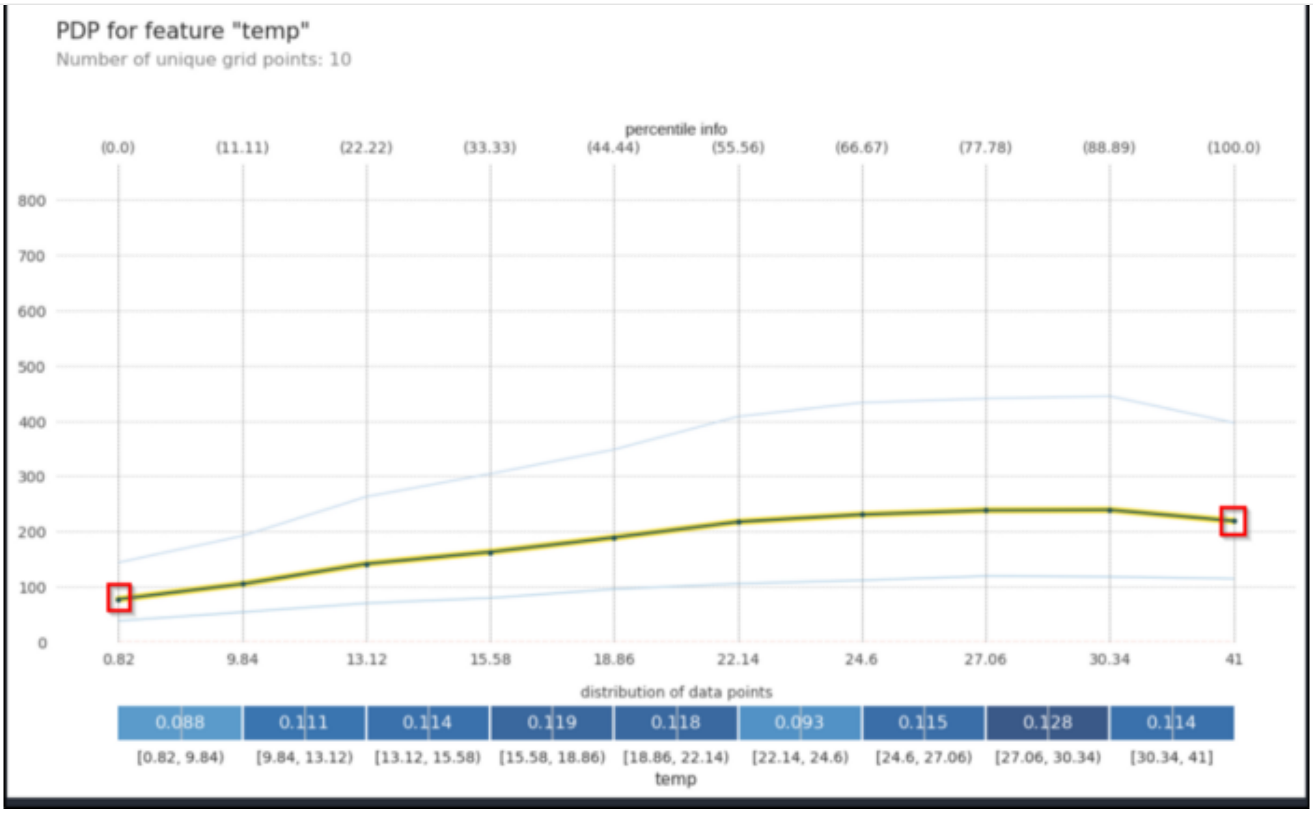
Along with the PDP plot(shown below in thick line), we take two **clustered** version of ICE plots. The two thin **blue** lines represent various ICE plots(corresponding to the marginal effect of 'temp' above-average and below-average respectively). The PDP starts from an average value of **~77**(sale of rentals). With increasing temperature('temp') there is an increase in sales of rentals consistently until the 'temp' value is **~30.** After this point, sales decline and end up being around **~217** on average. The corresponding code and plots follow.

```
#Drop the correlated variable 'atemp' and then fit
X,y = 
df[['year','hour','temp','humidity','windspeed','holiday','workingday
']],df['count']
X_train, X_test, y_train, y_test = train_test_split(X, y, 
test_size=0.25, random_state=42)

model = RandomForestRegressor(n_estimators=200) #Fit a Random Forest 
with 2oo Trees
visualizer = ResidualsPlot(model, qqplot=True) #Instantiate a 
Residual Plot Class

visualizer.fit(X_train, y_train)  # Fit the training data to the 
visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test 
data
visualizer.show()
```



Goodness of Fit(Regression with **No** Dependent Variables)(Source: Author)

```
pdp_temp = pdp.pdp_isolate(
    model=model, dataset=X_train, model_features=X_train.columns,
feature='temp'
)
fig, axes = pdp.pdp_plot(pdp_temp, 'temp',center=False,
cluster=True,n_cluster_centers=2,\
                         plot_lines=True, x_quantile=True,
show_percentile=True, plot_pts_dist=True )
```

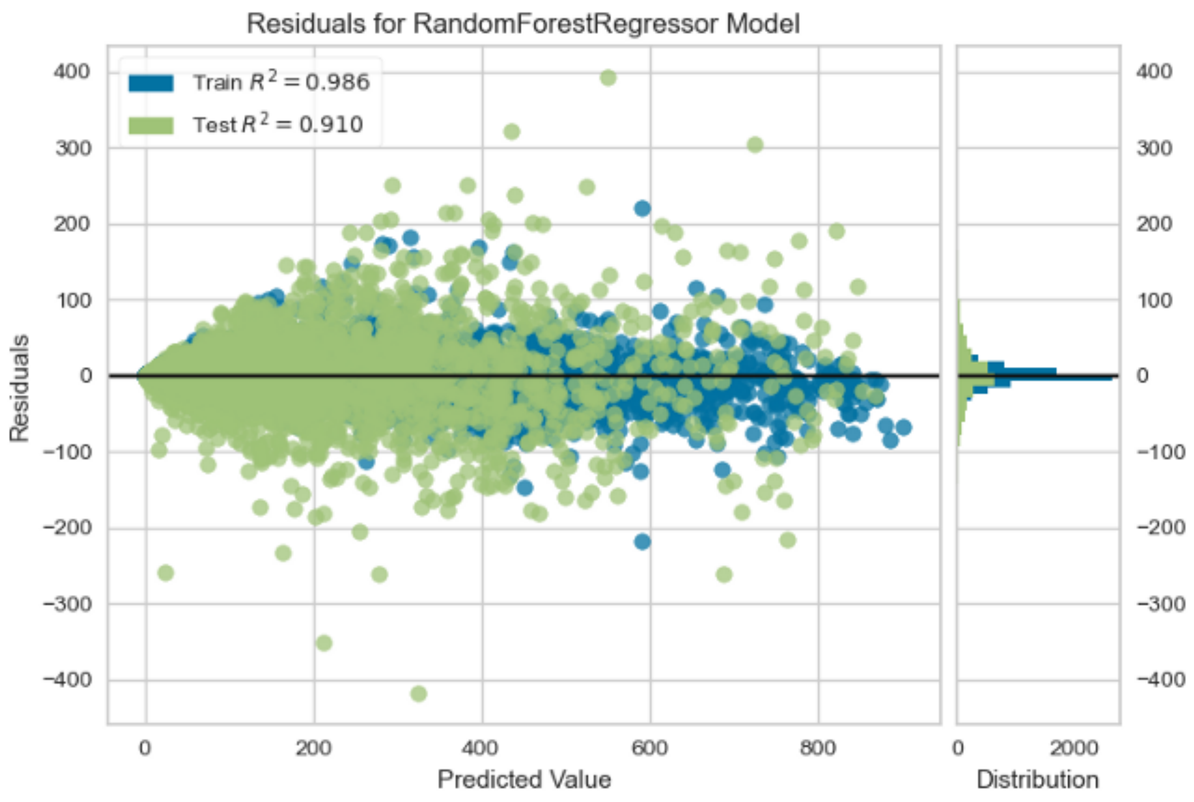


Partial Dependence Plot(with **no** dependent variables). The left red square is at ~77 and the right one is at ~217 marking the end of PDP(Source: Author)

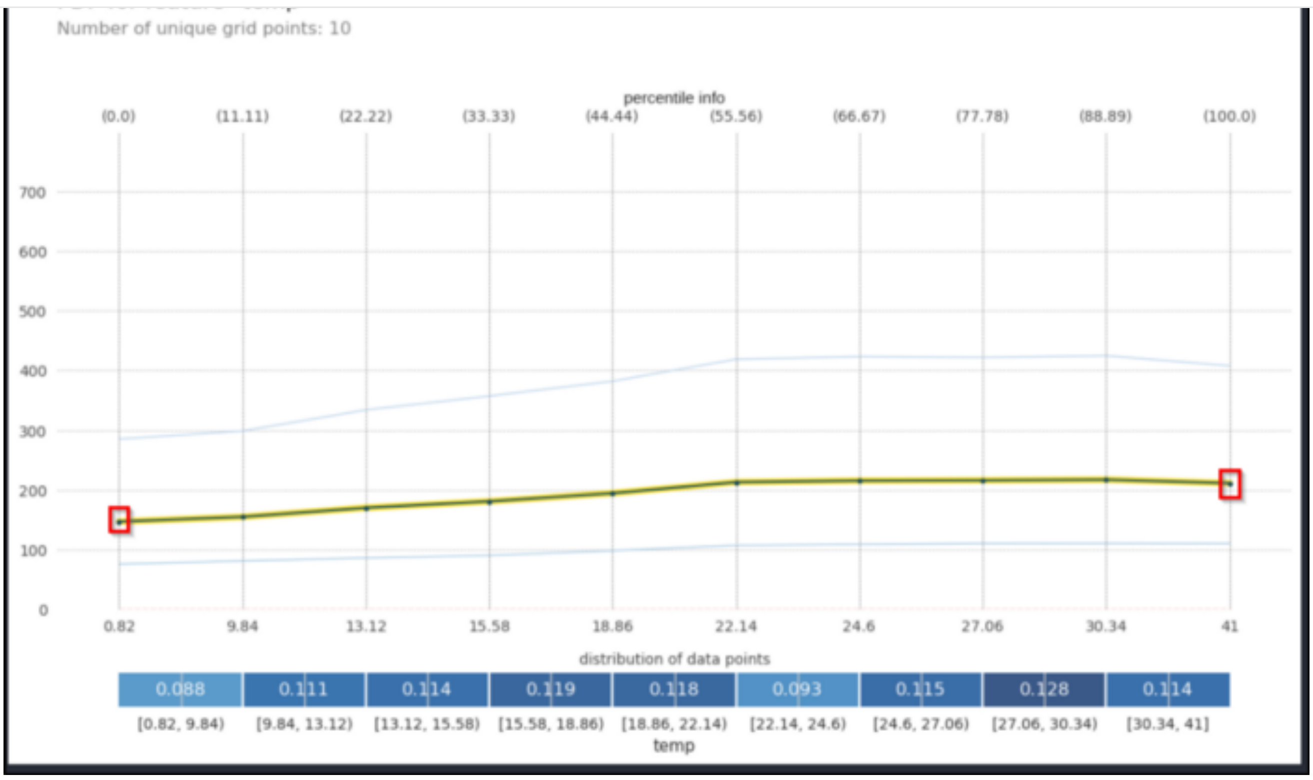
Partial Dependence Plot(with correlated variables)

In this case, we consider two continuous variables which are highly correlated — 'temp' and 'atemp' both as regressors. The PDP looks flatter somehow, isn't it? The PDP starts from a value of **~146**(average sales of rentals). This is in contrast with the above one which starts at **~77**. The plot ends this time at around a value of **~211**. **Why are the two plots different?**

```
#Include the correlated variable 'atemp' and then fit
X,y =
df[['year','hour','temp','atemp','humidity','windspeed','holiday','wo
rkingday']],df['count']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)

model = RandomForestRegressor(n_estimators=200)
visualizer = ResidualsPlot(model, qqplot=True)

visualizer.fit(X_train, y_train)  # Fit the training data to the
visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test
data
visualizer.show()
```



Goodness of Fit(Regression with Dependent Variables)(Source: Author)

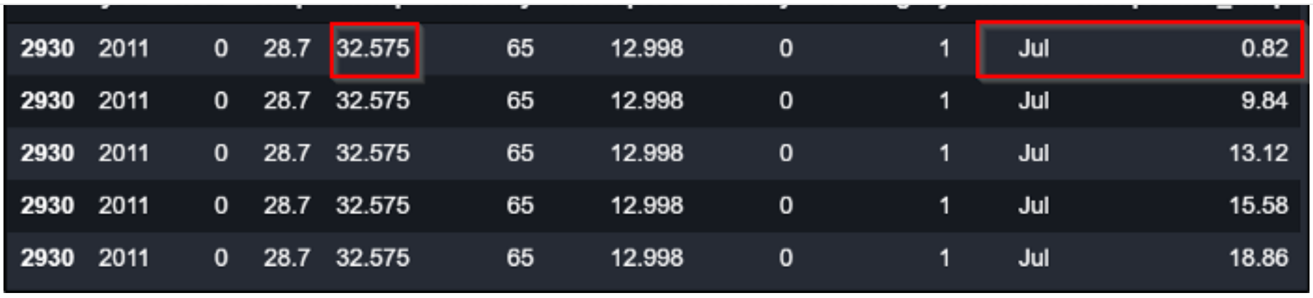


Partial Dependence Plot(with Dependent Variables). The left red square is at ~146 and the right one is at ~211 marking the end of PDP(Source: Author)

Pitfall to Avoid

PDP works with perturbation, i.e. for each data point, we extrapolate the values(here replacement in percentiles) of the feature of interest. Correlated variables result in spurious data points. For e.g in the following table, we see an extrapolated value of **0.82** in the month of July along with an 'atemp' value **32.5**(highlighted in red boxes on the table). Such a data point is infeasible(Washington D.C temperature in July can never be 0.82). So the effect of each variable smoothens out and we see spurious PDPs(a flatter plot in our case). **The solution -** to this is to remove one of the correlated variables('temp' or 'atemp') before fitting(in this case using Pearson Correlation).

```
X_train['month'] = df[df.index.isin(X_train.index)]['month']
X_train['month'] = X_train['month'].apply(lambda x:
calendar.month_abbr[x]) #Get the month
#Get all the possible values the feature temp(based on the quantile
grids) is allowed to take w.r.t
#to all the data points and values of other features reamining
constant.
X_train['extrapolated_temp'] =
[pdp_temp.feature_grids.tolist()]*len(X_train)
X_train = X_train.explode('extrapolated_temp')
X_train.head()
```



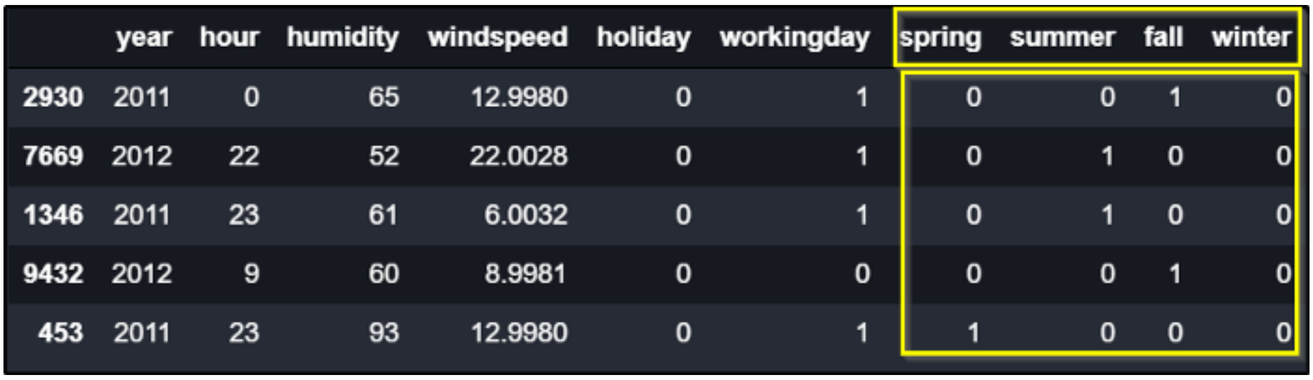
2930	2011	0	28.7	32.575	65	12.998	0	1	Jul	0.82
2930	2011	0	28.7	32.575	65	12.998	0	1	Jul	9.84
2930	2011	0	28.7	32.575	65	12.998	0	1	Jul	13.12
2930	2011	0	28.7	32.575	65	12.998	0	1	Jul	15.58
2930	2011	0	28.7	32.575	65	12.998	0	1	Jul	18.86

An extrapolated value of the temperature of 0.82 in the month of July(Source: Author)

Partial Dependence Plot — Categorical Case(no correlated variables)

Now, we remove 'temp'(as well as 'atemp') and instead we take 'season' as a regressor. Since the season is a categorical variable we take a one-hot encoded version of it. We evaluate the goodness of fit and get the PDP. What the PDP reveals is that with 'season' being either Spring or Winter the number of rental sales on an average is 118.8 and 201.476 respectively. The sales are maximum during Summers and Falls(217 and 230.4 on average respectively).

```
#Removing the variable 'temp' and instead using the variable 'season'
X,y =
df[['year','hour','season','humidity','windspeed','holiday','workingd
ay']],df['count']
X = pd.get_dummies(X,columns=['season']) #One hot encode
X = X.rename(columns = {'season_1':'spring','season_2':'summer',
                       'season_3':'fall','season_4':'winter'})
#Proper Renaming of Dummies
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
X_train.head()
```

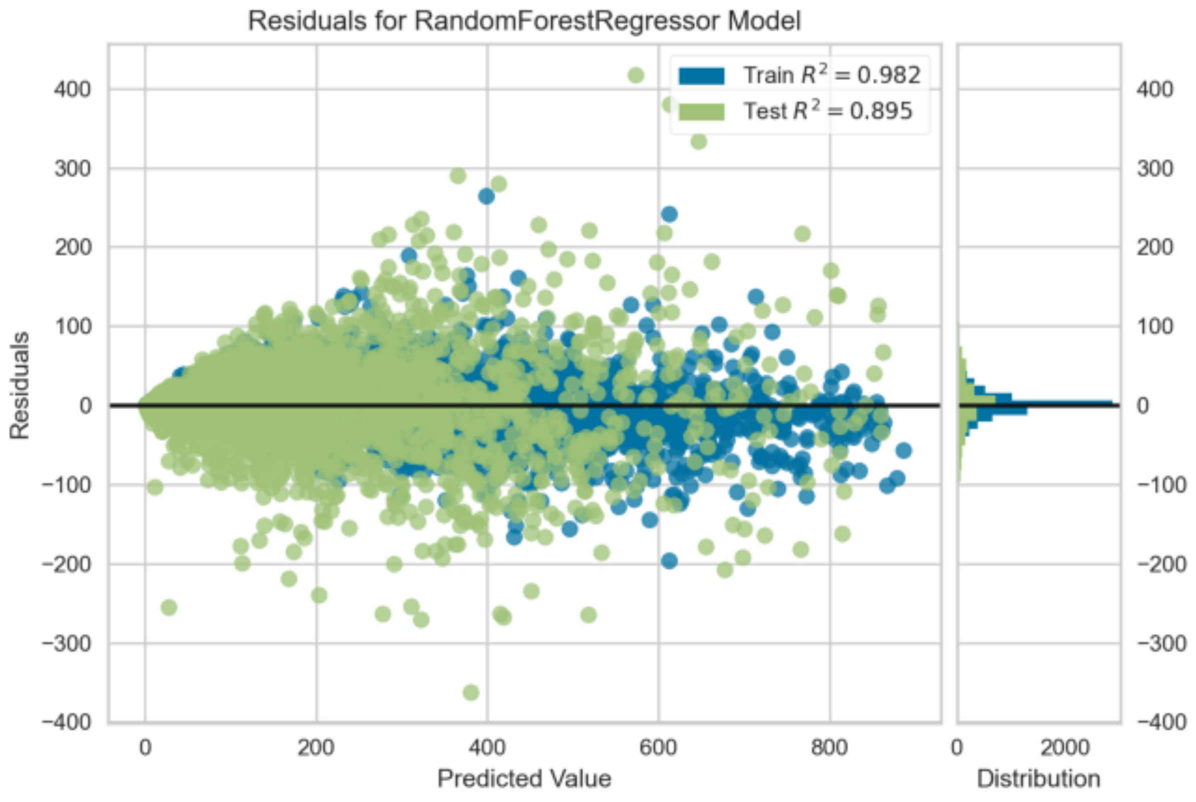


	year	hour	humidity	windspeed	holiday	workingday	spring	summer	fall	winter
2930	2011	0	65	12.9980	0	1	0	0	1	0
7669	2012	22	52	22.0028	0	1	0	1	0	0
1346	2011	23	61	6.0032	0	1	0	1	0	0
9432	2012	9	60	8.9981	0	0	0	0	1	0
453	2011	23	93	12.9980	0	1	1	0	0	0

One Hot Encoded Variable 'season'(Source: Author)

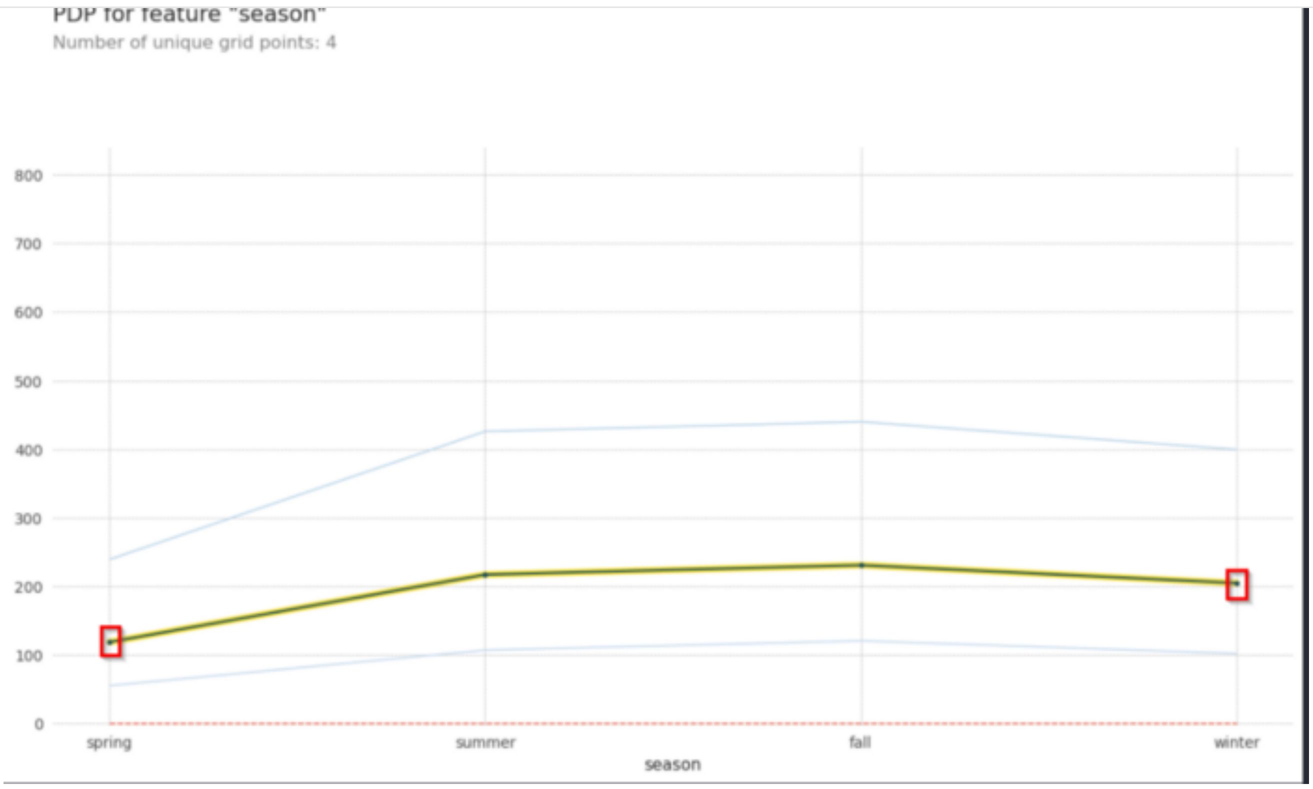
Get started Open in app

```
model = RandomForestRegressor(n_estimators=200)
visualizer = ResidualsPlot(model, qqplot=True)

visualizer.fit(X_train, y_train)  # Fit the training data to the
visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test
data
visualizer.show()
```



Goodness of Fit — Categorical Case(Regression with **No** Dependent Variables)(Source: Author)

```
pdp_season = pdp.pdp_isolate(
    model=model, dataset=X_train, model_features=X_train.columns,
    feature=['spring', 'summer', 'fall', 'winter']
)
fig, axes = pdp.pdp_plot(pdp_season,'season', center=False,
cluster=True,n_cluster_centers=2,\
                         plot_lines=True, x_quantile=True,
show_percentile=True)
```

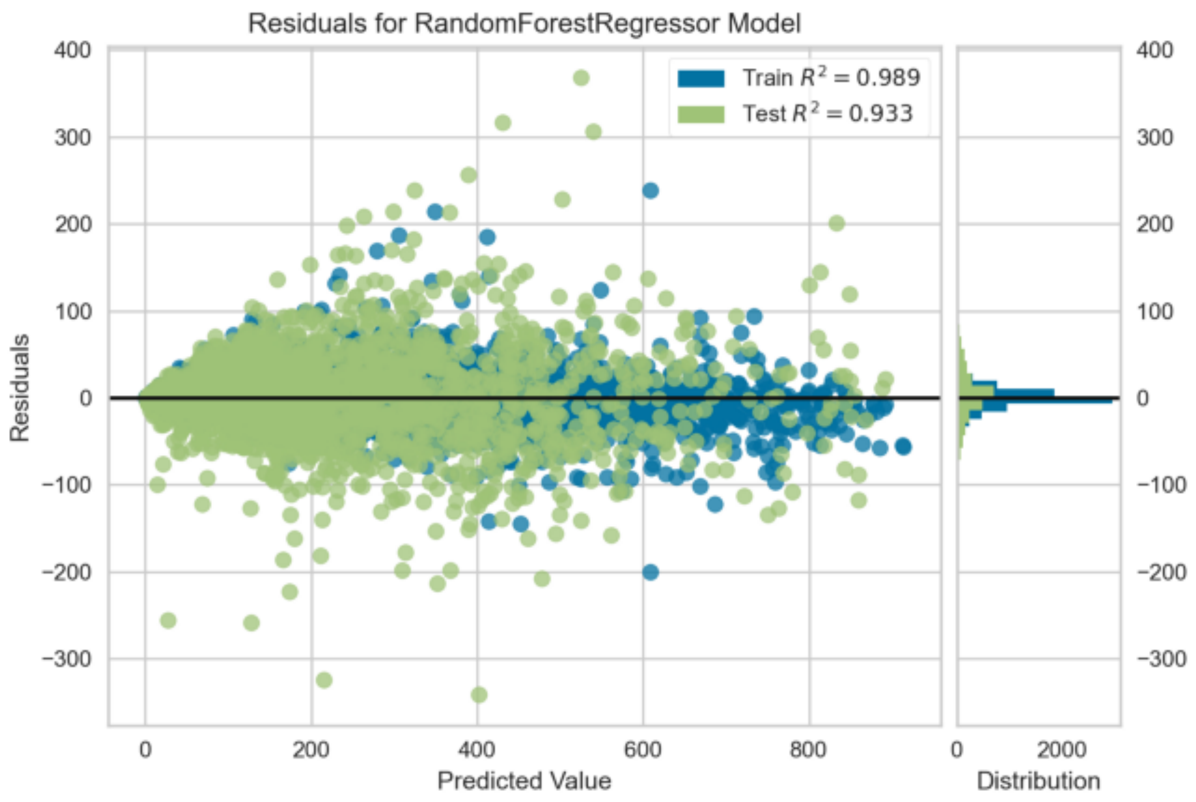


Partial Dependence Plot(with Dependent Variables). 118.5, 217.1, 230.5, 204.7 PDP values for spring, summer, fall, and winter respectively. (Source: Author)

Partial Dependence Plot — Categorical Case(with correlated variables)

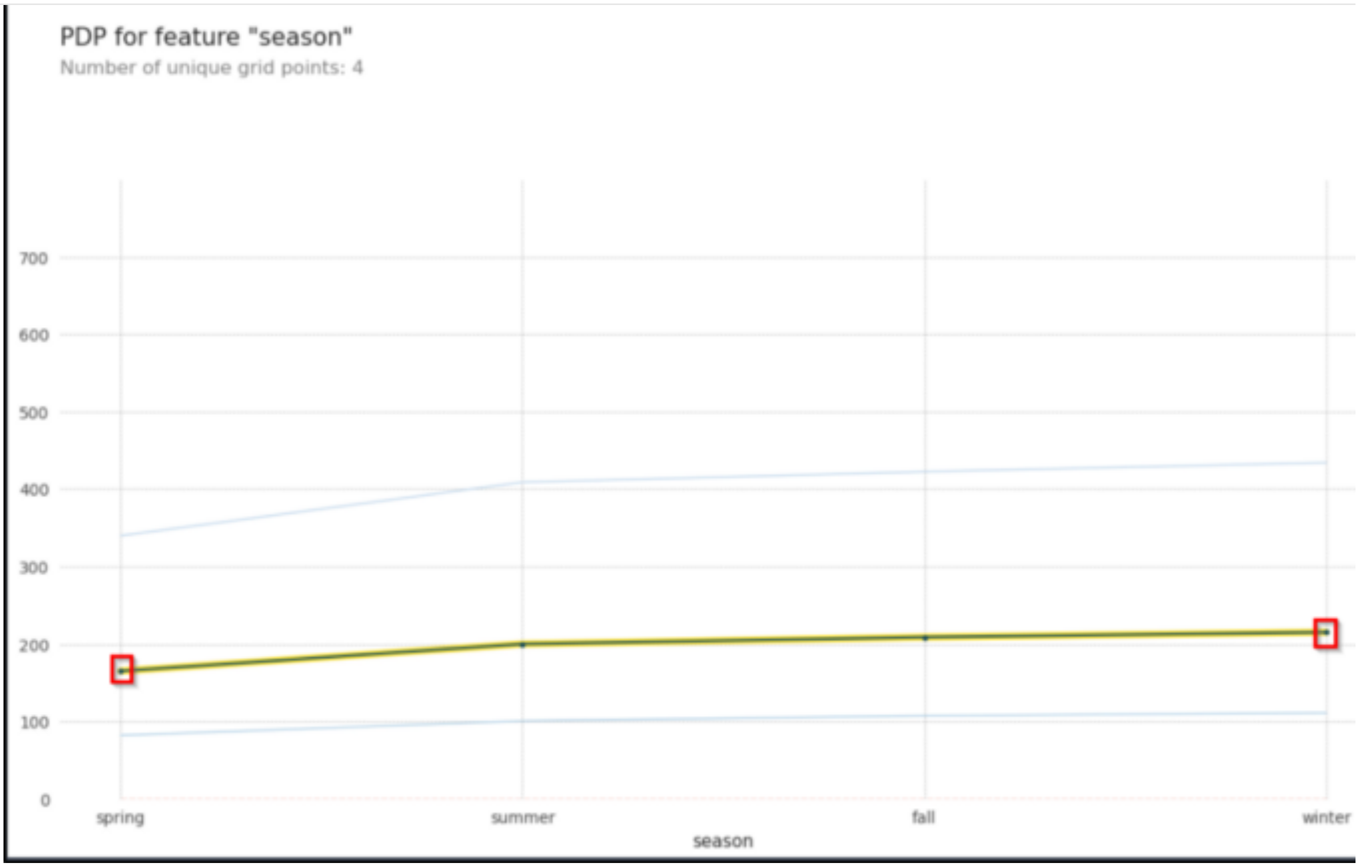
From our ANOVA table, we know that 'season' and 'temp' are dependent, so now along with 'season' we include 'temp' also. We evaluate the goodness of fit and eventually the PDP. We see very similar results, the PDP in this case is again a lot flatter compared to the one with no correlated variables.

```
#Included the dependent variable 'temp'
X,y = 
df[['year','hour','temp','season','humidity','windspeed','holiday','w
orkingday']],df['count']
X = pd.get_dummies(X,columns=['season'])
X = X.rename(columns = {'season_0':'spring','season_1':'summer',
                       'season_2':'fall','season_3':'winter'})
X_train, X_test, y_train, y_test = train_test_split(X, y, 
test_size=0.25, random_state=42)

#Fit the model
model = RandomForestRegressor(n_estimators=200)
visualizer = ResidualsPlot(model, qqplot=True)

visualizer.fit(X_train, y_train)  # Fit the training data to the 
visualizer
visualizer.score(X_test, y_test)  # Evaluate the model on the test
```



Goodness of Fit — Categorical Case(Regression with Dependent Variables)(Source: Author)

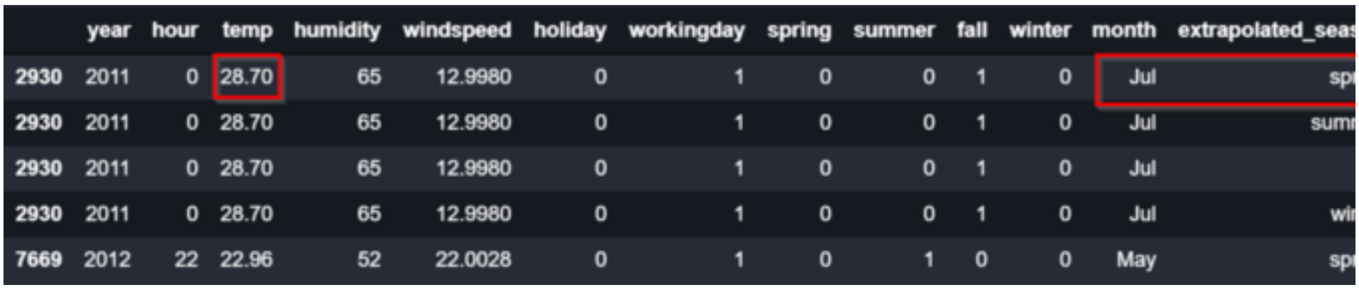
```
pdp_season = pdp.pdp_isolate(
    model=model, dataset=X_train, model_features=X_train.columns,
    feature=['spring', 'summer', 'fall', 'winter']
)
fig, axes = pdp.pdp_plot(pdp_season,'season', center=False,
cluster=True,n_cluster_centers=2,\
                         plot_lines=True, x_quantile=True,
show_percentile=True)
```



Partial Dependence Plot(with Dependent Variables). 165.6, 199.6, 208.6, 215.2 PDP values for spring, summer, fall, a winter respectively. (Source: Author)

Since 'season' and 'temp' are dependent we see that effects of both the features smoothen ou when both are used as regressors. This is the reason that the second PDP is a lot flatter. Dwelling deeper, we observe the extrapolated values of 'season'. The first row for e.g shows that for the month of 'July' the corresponding extrapolated season is 'Spring'(highlighted in red in the table). This is an infeasible data point and hence produces spurious results as the marginal effects of both the dependent variables cancel out and the plot is flat. **The solution** we have to check for continuous-categorical dependencies and remove one of them from ou list of regressors(using **ANOVA**).

```
X_train['month'] = df[df.index.isin(X_train.index)]['month'] #Get the
Month
X_train['month'] = X_train['month'].apply(lambda x:
calendar.month_abbr[x])#Get the Month
#Get all the possible values the feature season is allowed to take w.r.t
to all
#the data points and values of other features reamining constant.
X_train['extrapolated_season'] =
[pdp_season.feature_grids.tolist()]*len(X_train)
```



	year	hour	temp	humidity	windspeed	holiday	workingday	spring	summer	fall	winter	month	extrapolated_seas
2930	2011	0	28.70	65	12.9980	0	1	0	0	1	0	Jul	sp
2930	2011	0	28.70	65	12.9980	0	1	0	0	1	0	Jul	sum
2930	2011	0	28.70	65	12.9980	0	1	0	0	1	0	Jul	
2930	2011	0	28.70	65	12.9980	0	1	0	0	1	0	Jul	wi
7669	2012	22	22.96	52	22.0028	0	1	0	1	0	0	May	sp

An extrapolated season of 'spring' in the month of July(Source: Author)

Conclusion

Feature Dependence is not only about Correlation. We can make further checks like Categorical to Categorical relationships using **Chi-Square** tests, and also **Non-Linear** Featu Dependency tests(ones based on **Kernel methods** for e.g). The same deductions as above al apply for **ICE** plots.

We show in this article how PDPs and ICE plots can be misinterpreted because of dependencies.

References

1. Christoph Molnar, Gunnar König, Julia Herbinger, Timo Freiesleben, Susanne Dandl, Christian A. Scholbeck, Giuseppe Casalicchio, Moritz Grosse-Wentrup, Bernd Bischl - https://arxiv.org/abs/2007.04131

2. https://christophm.github.io/interpretable-ml-book/

3. https://compstat-lmu.github.io/iml_methods_limitations/pdp-correlated.html

Libraries/Data Used

1. https://www.kaggle.com/c/bike-sharing-demand/data

2. https://www.scikit-yb.org/en/latest/

3. https://www.statsmodels.org/stable/generated /statsmodels.formula.api.ols.html#statsmodels.formula.api.ols

4. https://scikit-learn.org/stable/modules/generated /sklearn.ensemble.RandomForestClassifier.html

5. https://pdpbox.readthedocs.io/en/latest/