Feature Importance

Lesson Objectives

By the end of this lesson, students will be able to:

- Explain what feature importance is, which models use it, and what it means.
- Visualize and construct recommendations using feature importance.
- Implement scikit-learn's permutation_importance, and its advantages over built-in importance.

Previously....

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns


## Reviewing the options used
pd.set_option('display.max_columns',100)
pd.set_option('display.max_rows',100)
pd.set_option('display.float_format', lambda x: f"{x:,.2f}")

## Customization Options
plt.style.use(['fivethirtyeight','seaborn-talk'])
mpl.rcParams['figure.facecolor']='white'

## additional required imports
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import make_column_transformer, make_column_selector, Colu
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn import metrics
from sklearn.linear_model import LinearRegression
import joblib

SEED = 321
np.random.seed(SEED)
```

Code/Model From Previous Lesson

In [2]:
```
## Load in the King's County housing dataset and display the head and info
url = "https://docs.google.com/spreadsheets/d/e/2PACX-1vS6xDKNpWkBBdhZSqepy48bX

df = pd.read_excel(url,sheet_name='student-mat')
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395 entries, 0 to 394
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   school      395 non-null    object 
 1   sex         395 non-null    object 
 2   age         395 non-null    float64
 3   address     395 non-null    object 
 4   famsize     395 non-null    object 
 5   Pstatus     395 non-null    object 
 6   Medu        395 non-null    float64
 7   Fedu        395 non-null    float64
 8   Mjob        395 non-null    object 
 9   Fjob        395 non-null    object 
 10  reason      395 non-null    object 
 11  guardian    395 non-null    object 
 12  traveltime  395 non-null    float64
 13  studytime   395 non-null    float64
 14  failures    395 non-null    float64
 15  schoolsup   395 non-null    object 
 16  famsup      395 non-null    object 
 17  paid        395 non-null    object 
 18  activities  395 non-null    object 
 19  nursery     395 non-null    object 
 20  higher      395 non-null    object 
 21  internet    395 non-null    object 
 22  romantic    395 non-null    object 
 23  famrel      395 non-null    float64
 24  freetime    395 non-null    float64
 25  goout       395 non-null    float64
 26  Dalc        395 non-null    float64
 27  Walc        395 non-null    float64
 28  health      395 non-null    float64
 29  absences    395 non-null    float64
 30  G1          395 non-null    float64
 31  G2          395 non-null    float64
 32  G3          395 non-null    float64
dtypes: float64(16), object(17)
memory usage: 102.0+ KB
```

Out[2]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	gua
0	GP	F	18.00	U	GT3	A	4.00	4.00	at_home	teacher	course	m
1	GP	F	17.00	U	GT3	T	1.00	1.00	at_home	other	course	
2	GP	F	15.00	U	LE3	T	1.00	1.00	at_home	other	other	m
3	GP	F	15.00	U	GT3	T	4.00	2.00	health	services	home	m
4	GP	F	16.00	U	GT3	T	3.00	3.00	other	other	home	

In [3]:

```
# ### Train Test Split
## Make x and y variables
y = df['G3'].copy()
X = df.drop(columns=['G3']).copy()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```python
## train-test-split with random state for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=SEED)


# ### Preprocessing + ColumnTransformer

## make categorical & numeric selectors
cat_sel = make_column_selector(dtype_include='object')
num_sel = make_column_selector(dtype_include='number')

## make pipelines for categorical vs numeric data
cat_pipe = make_pipeline(SimpleImputer(strategy='constant',
                                       fill_value='MISSING'),
                         OneHotEncoder(drop='if_binary', sparse=False))

num_pipe = make_pipeline(SimpleImputer(strategy='mean'))

## make the preprocessing column transformer
preprocessor = make_column_transformer((num_pipe, num_sel),
                                       (cat_pipe,cat_sel),
                                      verbose_feature_names_out=False)

## fit column transformer and run get_feature_names_out
preprocessor.fit(X_train)
feature_names = preprocessor.get_feature_names_out()

X_train_df = pd.DataFrame(preprocessor.transform(X_train),
                          columns = feature_names, index = X_train.index)


X_test_df = pd.DataFrame(preprocessor.transform(X_test),
                          columns = feature_names, index = X_test.index)
X_test_df.head(3)
```

Out[3]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc
58	15.00	1.00	2.00	1.00	2.00	0.00	4.00	3.00	2.00	1.00	1.00
338	18.00	3.00	3.00	1.00	4.00	0.00	5.00	3.00	3.00	1.00	1.00
291	17.00	4.00	3.00	1.00	3.00	0.00	4.00	2.00	2.00	1.00	2.00

In [4]:

```python
def evaluate_linreg(model, X_train,y_train, X_test,y_test, get_coeffs=True):
    from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_err

    results = []
    y_hat_train = model.predict(X_train)
    r2_train = r2_score(y_train,y_hat_train)
    rmse_train = mean_squared_error(y_train,y_hat_train, squared=False)
    results.append({'Data':'Train', 'R^2':r2_train, "RMSE": rmse_train})

    y_hat_test = model.predict(X_test)
    r2_test = r2_score(y_test,y_hat_test)
    rmse_test = mean_squared_error(y_test,y_hat_test, squared=False)
    results.append({'Data':'Test', 'R^2':r2_test, "RMSE": rmse_test})

    results_df = pd.DataFrame(results).round(3).set_index('Data')
    results_df.index.name=None
    print(results_df)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
    if get_coeffs:
        try:
            coeffs = pd.Series(model.coef_, index= X_train.columns)
            coeffs.loc['intercept'] = model.intercept_
            return coeffs
        except:
            print('[!] Could not extract coefficients from model.')
```

"Best" Model From Previous Lesson

Modeling with Ordinal Zipcodes

- We will need to remake our X,y, X_train,y_train, etc.
- We will also want to make a copy of our preprocessor so we can leave the original one intact with the values it learned from the earlier X/y data.

In [5]:
```
## fitting a linear regression model
lin_reg = LinearRegression(fit_intercept=True)
lin_reg.fit(X_train_df, y_train)
coeffs_orig = evaluate_linreg(lin_reg, X_train_df, y_train, X_test_df,y_test)
coeffs_orig
```

```
       R^2  RMSE
Train 0.85  1.83
Test  0.81  1.85
```

Out[5]:
```
age                -0.22
Medu                0.29
Fedu               -0.18
traveltime          0.19
studytime          -0.22
failures           -0.10
famrel              0.31
freetime            0.02
goout              -0.02
Dalc               -0.21
Walc                0.26
health              0.03
absences            0.05
G1                  0.14
G2                  0.99
school_MS           0.38
sex_M              -0.01
address_U           0.15
famsize_LE3         0.01
Pstatus_T           0.38
Mjob_at_home       -0.05
Mjob_health        -0.02
Mjob_other          0.16
Mjob_services       0.12
Mjob_teacher       -0.21
Fjob_at_home        0.33
Fjob_health         0.38
Fjob_other         -0.11
Fjob_services      -0.54
```

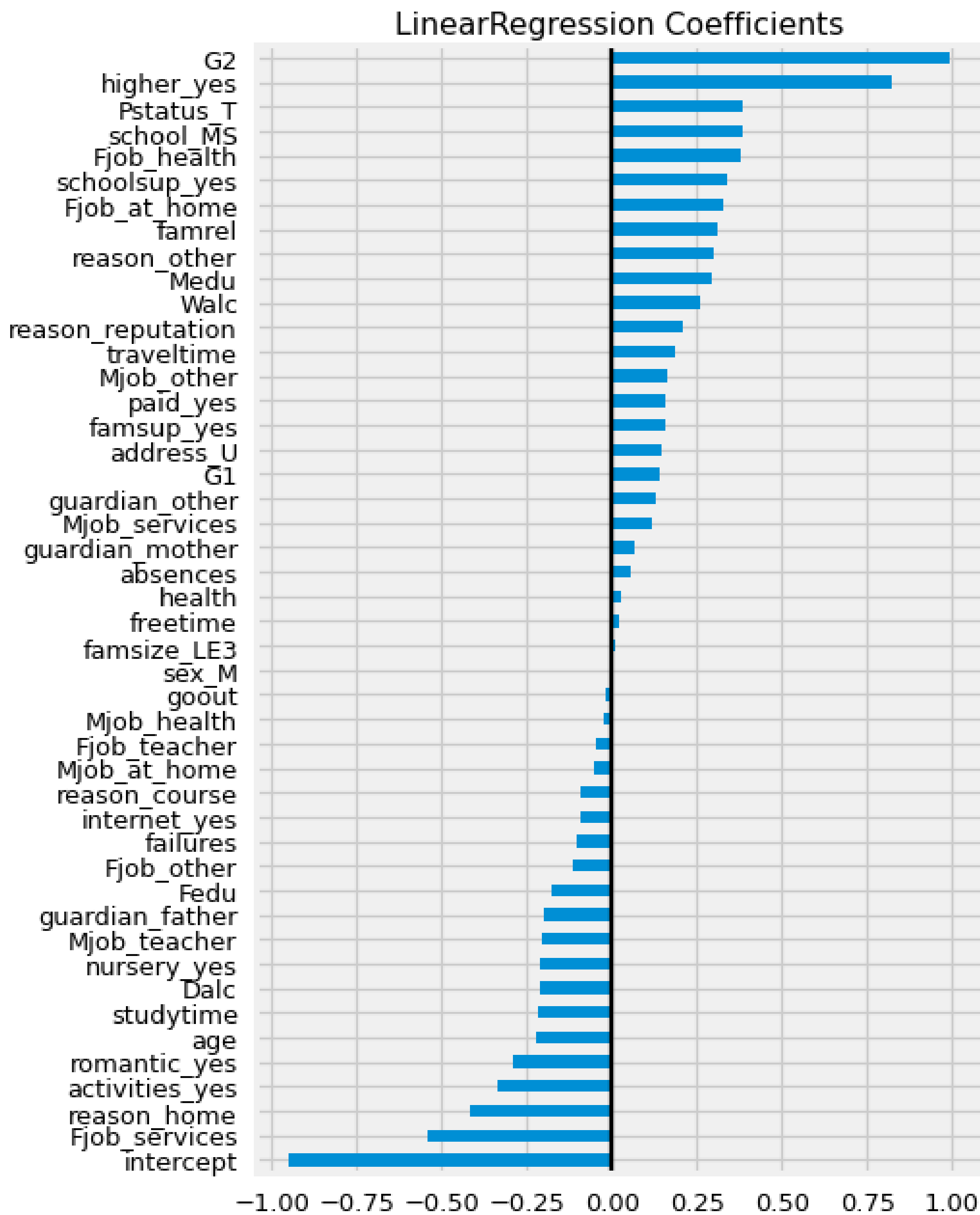
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
reason_course       -0.09
reason_home         -0.42
reason_other         0.30
reason_reputation    0.21
guardian_father     -0.20
guardian_mother      0.07
guardian_other       0.13
schoolsup_yes        0.34
famsup_yes           0.16
paid_yes             0.16
activities_yes      -0.34
nursery_yes         -0.21
higher_yes           0.82
internet_yes        -0.09
romantic_yes        -0.29
intercept           -0.95
dtype: float64
```

In [6]:

```
ax = coeffs_orig.sort_values().plot(kind='barh',figsize=(6,10))
ax.axvline(0,color='k')
ax.set_title('LinearRegression Coefficients');
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Tree Based Models - Feature Importance

- There are many models that do not use/calculate coefficients as part of the modeling process.
- Tree-Based models (decision trees, random forests, xgboost, etc) use/calculate feature importance.

- According to the Scikit-Learn RandomForest Documentation on Feature Importance "**Feature Importance**"is:
  - "The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance."
    - In other words, its how helpful each feature was in growing/sorting the tree- ful a feature was in separating the data. The more

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

"important" it is.

In [7]:
```
## fit random forest
from sklearn.ensemble import RandomForestRegressor
rf_reg = RandomForestRegressor()
rf_reg.fit(X_train_df,y_train)
evaluate_linreg(rf_reg,X_train_df,y_train,X_test_df,y_test)
```

```
       R^2  RMSE
Train 0.98  0.66
Test  0.91  1.24
[!] Could not extract coefficients from model.
```

In [8]:
```
# get importance
def get_importance(model, feature_names):
    df_importance = pd.Series(model.feature_importances_,
                              index=feature_names)
    return df_importance.sort_values(ascending=False)

importances = get_importance(rf_reg, feature_names)
importances
```

Out[8]:
```
G2                    0.77
absences              0.12
age                   0.02
studytime             0.01
health                0.01
G1                    0.01
famrel                0.01
schoolsup_yes         0.01
Medu                  0.00
reason_home           0.00
activities_yes        0.00
Fedu                  0.00
Walc                  0.00
goout                 0.00
reason_reputation     0.00
school_MS             0.00
failures              0.00
freetime              0.00
traveltime            0.00
paid_yes              0.00
guardian_mother       0.00
reason_course         0.00
Fjob_services         0.00
famsize_LE3           0.00
Mjob_teacher          0.00
Fjob_other            0.00
Mjob_services         0.00
sex_M                 0.00
Mjob_other            0.00
internet_yes          0.00
nursery_yes           0.00
romantic_yes          0.00
guardian_father       0.00
Fjob_health           0.00
famsup_yes            0.00
Dalc                  0.00
Mjob_at_home          0.00
```

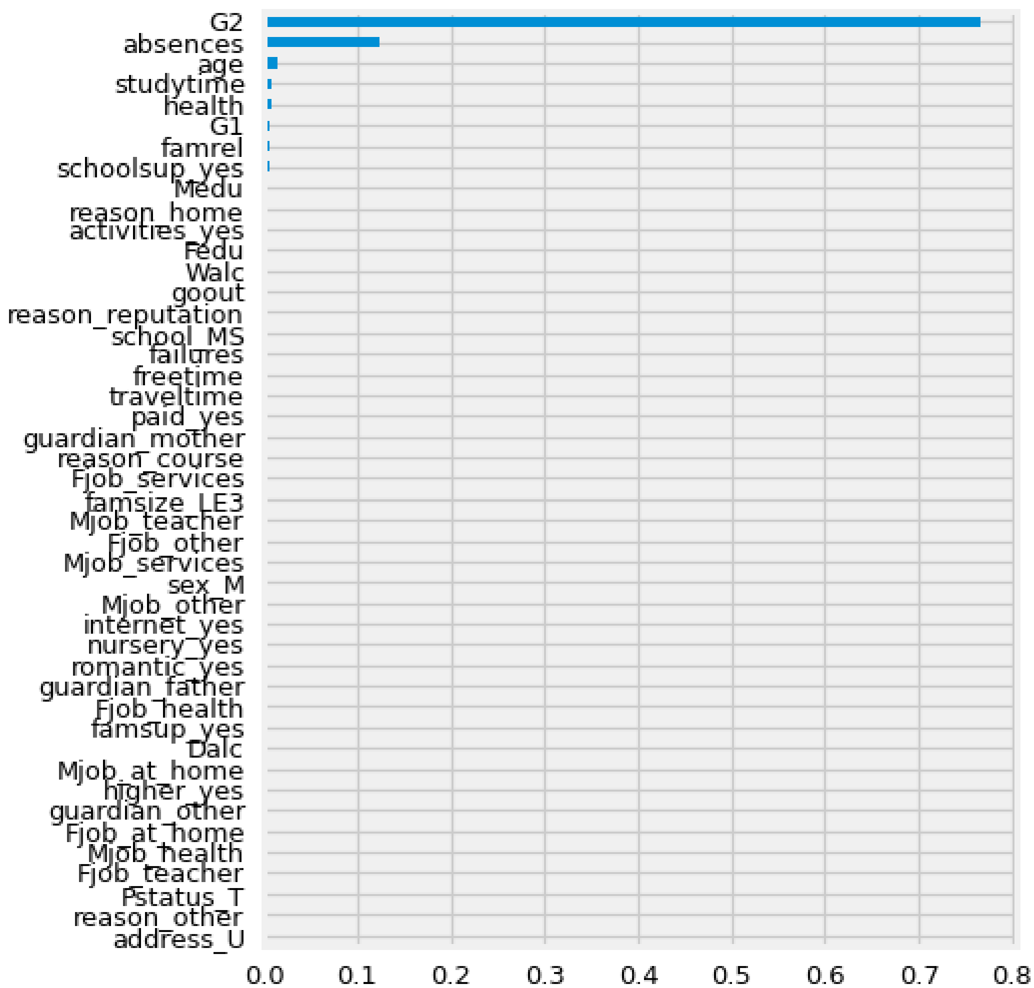
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
guardian_other      0.00
Fjob_at_home        0.00
Mjob_health         0.00
Fjob_teacher        0.00
Pstatus_T           0.00
reason_other        0.00
address_U           0.00
dtype: float64
```

Interpreting Feature Importance

In [9]:
```python
# plot importance
importances.sort_values().plot(kind='barh',figsize=(6,8))
```

Out[9]: <AxesSubplot:>



- **What the feature importances tell us:**
  - G2 is by far the single most important feature for predicting G3.
  - The # of absences is the second most important.
  - Everything is relatively unimportant.
- **What the feature importances don't tell us:**
  - Notice that all of the values on the graph are positive.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- There is no +/- directionality with feature importance!
  - We only know that a feature was heavily used to predict the target, but we DON'T KNOW the actual **relationship** between the feature and target.
  - Does having a higher G2 mean higher G3?
  - Does more absences mean a higher G3?

    > We don't know!

- **Additional Considerations/Caveats for Feature Importance**
  - Also from the Scikit-Learn RandomForest Documentation on Feature Importance:
  - " Warning: impurity-based feature importances can be misleading for high cardinality features (many unique values). See sklearn.inspection.permutation_importance as an alternative."
  - In other words, model-based feature importance is biased towards valuing features with many values (

Modeling without Confounding/Inappropriate Features Features

- If we weren't sure that we should remove G1 and G2 before, look at how much G2 is DOMINATING the feature importance. We can hardly see the other features.
  - Considering the business case considerations we discussed previously, this really confirms that including G2 is not going to help us get a better understanding of which students perform better in year 3 and why.

In [10]:
```
## Drop the G1 and G2 features from the x vars
X_train_df = X_train_df.drop(columns=['G1','G2'])
X_test_df = X_test_df.drop(columns=['G1','G2'])
X_train_df
```

Out[10]:

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc
215	17.00	3.00	2.00	2.00	2.00	0.00	4.00	4.00	4.00	1.00	3.00
48	15.00	4.00	2.00	1.00	2.00	0.00	4.00	3.00	3.00	2.00	2.00
303	17.00	3.00	2.00	1.00	4.00	0.00	5.00	2.00	2.00	1.00	2.00
160	17.00	2.00	1.00	2.00	1.00	2.00	3.00	3.00	2.00	2.00	2.00
60	16.00	4.00	4.00	1.00	2.00	0.00	2.00	4.00	4.00	2.00	3.00
...	...	...	...	...	...	...	...	...	...	...	...
200	16.00	4.00	3.00	1.00	2.00	0.00	4.00	3.00	5.00	1.00	5.00
297	18.00	4.00	3.00	2.00	2.00	0.00	4.00	4.00	5.00	1.00	2.00
287	17.00	1.00	1.00	1.00	3.00	0.00	4.00	3.00	3.00	1.00	1.00

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	age	Medu	Fedu	traveltime	studytime	failures	famrel	freetime	goout	Dalc	Walc
124	16.00	2.00	2.00	1.00	2.00	0.00	5.00	4.00	4.00	1.00	1.00
26	15.00	2.00	2.00	1.00	1.00	0.00	4.00	2.00	2.00	1.00	2.00

296 rows × 43 columns

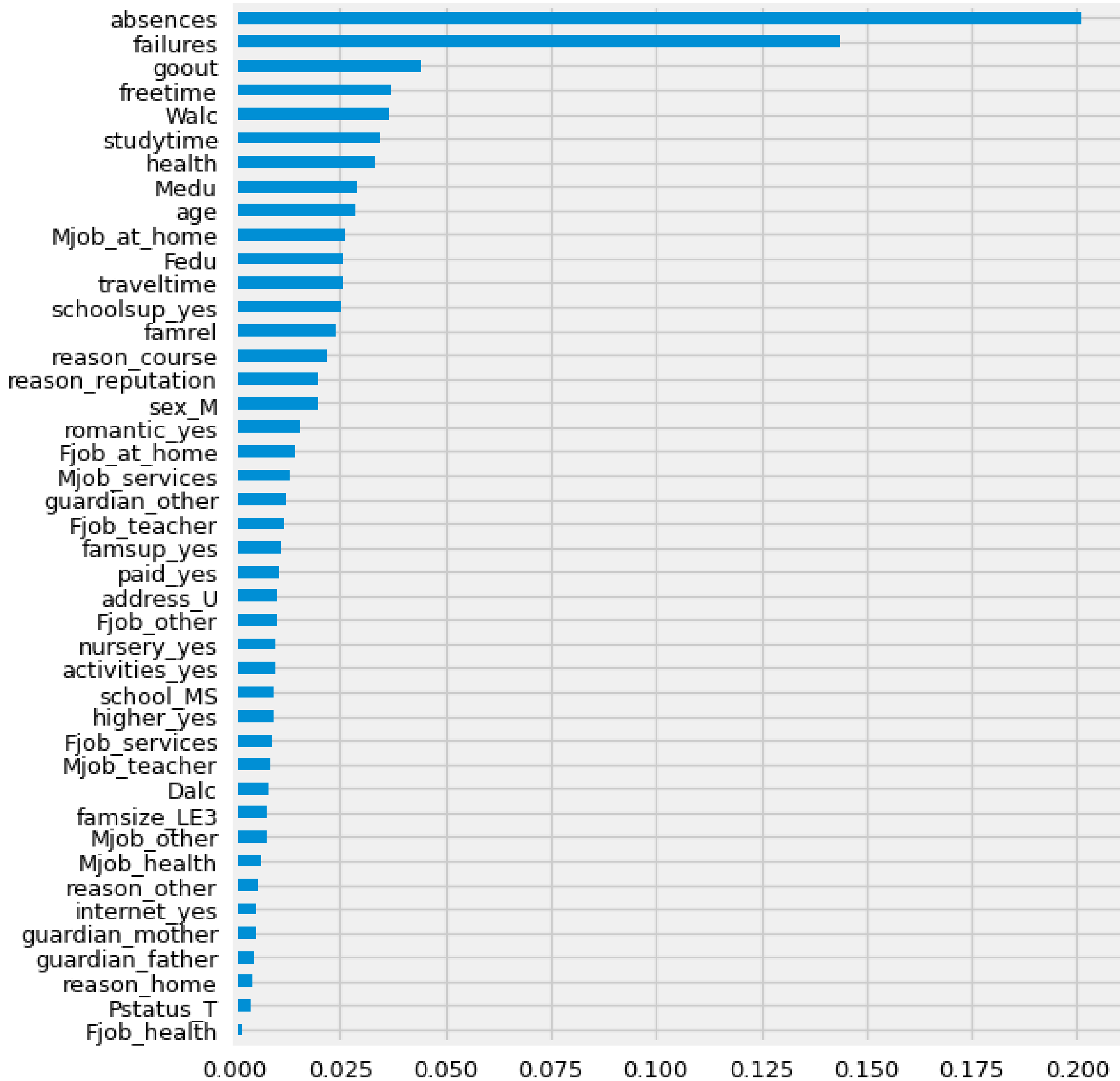
In [11]:
```
def evaluate_regression(model, X_train,y_train, X_test,y_test, get_params=True,
                        sort_params=True,ascending=True):
    from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_err

    results = []
    y_hat_train = model.predict(X_train)
    r2_train = r2_score(y_train,y_hat_train)
    rmse_train = mean_squared_error(y_train,y_hat_train, squared=False)
    results.append({'Data':'Train', 'R^2':r2_train, "RMSE": rmse_train})

    y_hat_test = model.predict(X_test)
    r2_test = r2_score(y_test,y_hat_test)
    rmse_test = mean_squared_error(y_test,y_hat_test, squared=False)
    results.append({'Data':'Test', 'R^2':r2_test, "RMSE": rmse_test})

    results_df = pd.DataFrame(results).round(3).set_index('Data')
    results_df.index.name=None
    print(results_df)

    if get_params:
        ## if a regression with coef_
        if hasattr(model, 'coef_'):
            params = pd.Series(model.coef_, index= X_train.columns,
                               name='Coefficients')
            params.loc['intercept'] = model.intercept_

        ## if a tree model with feature importance
        elif hasattr(model, 'feature_importances_'):
            params = pd.Series(model.feature_importances_,
                               index=X_train.columns, name='Feature Importance')

        else:
            print('[!] Could not extract coefficients or feature importances fr
    if sort_params:
        return params.sort_values(ascending=ascending)
    else:
        return params
```

In [12]:
```
rf_reg = RandomForestRegressor()
rf_reg.fit(X_train_df,y_train)
importances = evaluate_regression(rf_reg,X_train_df,y_train,X_test_df,y_test)
importances.plot(kind='barh',figsize=(8,10))
```

```
       R^2   RMSE
Train 0.91   1.44
Test  0.13   3.92
```

Out[12]:
```
<AxesSubplot:>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Interpreting Feature Importance

- TO DO: Interpret

Beyond Built-In Importances

- Scikit-learn has a tool called permutation importance that will calculate feature importance, without the issues discussed above.
- The function will take our model and our features and it will repeat the modeling for each feature.
    - One at a time, for each feature, it will shuffle all of the rows JUST IN THAT ONE FEATURE and repeat the modeling process.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- The idea is that we are scrambling/destroying that features relationship to our target.

- Then, it examines which feature caused the biggest decrease in the model's performance, and it uses this information to determine the "permutation importance"

In [13]:
```
from sklearn.inspection import permutation_importance
## Permutation importance takes a fit mode and test data.
r = permutation_importance(rf_reg, X_train_df, y_train,n_repeats =5)
r.keys()
```

Out[13]:
```
dict_keys(['importances_mean', 'importances_std', 'importances'])
```

In [14]:
```
## can make the mean importances into a series
permutation_importances = pd.Series(r['importances_mean'],index=X_train_df.colu
                           name = 'permutation importance')
permutation_importances
```

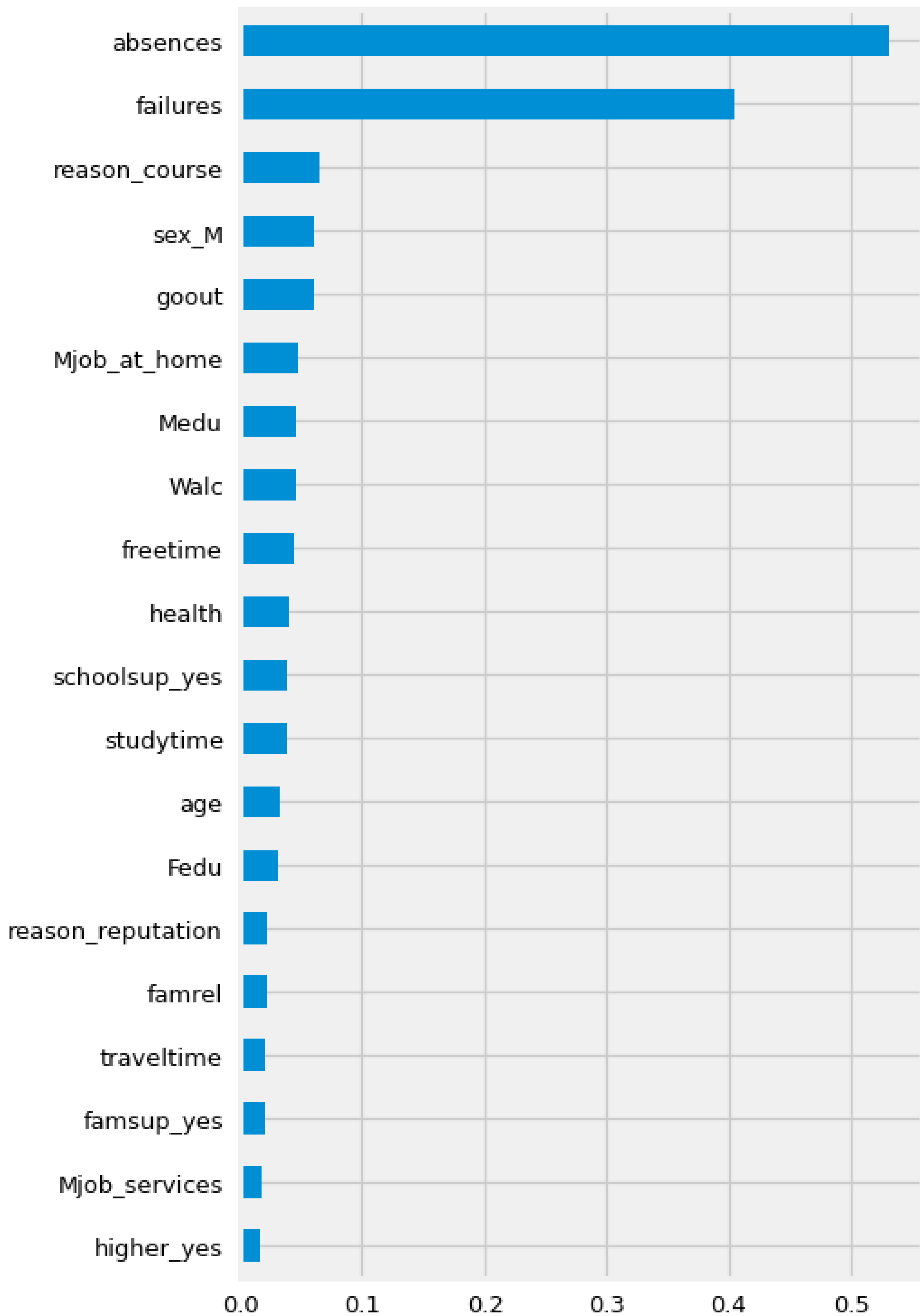
Out[14]:
```
age                  0.03
Medu                 0.05
Fedu                 0.03
traveltime           0.02
studytime            0.04
failures             0.41
famrel               0.02
freetime             0.04
goout                0.06
Dalc                 0.01
Walc                 0.05
health               0.04
absences             0.53
school_MS            0.01
sex_M                0.06
address_U            0.01
famsize_LE3          0.01
Pstatus_T            0.00
Mjob_at_home         0.05
Mjob_health          0.01
Mjob_other           0.01
Mjob_services        0.02
Mjob_teacher         0.01
Fjob_at_home         0.02
Fjob_health          0.00
Fjob_other           0.01
Fjob_services        0.01
Fjob_teacher         0.01
reason_course        0.07
reason_home          0.00
reason_other         0.01
reason_reputation    0.02
guardian_father      0.00
guardian_mother      0.00
guardian_other       0.01
schoolsup_yes        0.04
famsup_yes           0.02
paid_yes             0.01
activities_yes       0.01
nursery_yes          0.01
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
higher_yes          0.02
internet_yes        0.00
romantic_yes        0.02
Name: permutation importance, dtype: float64
```

In [15]:
```
permutation_importances.sort_values().tail(20).plot(kind='barh',figsize=(6,12))
```

Out[15]: <AxesSubplot:>

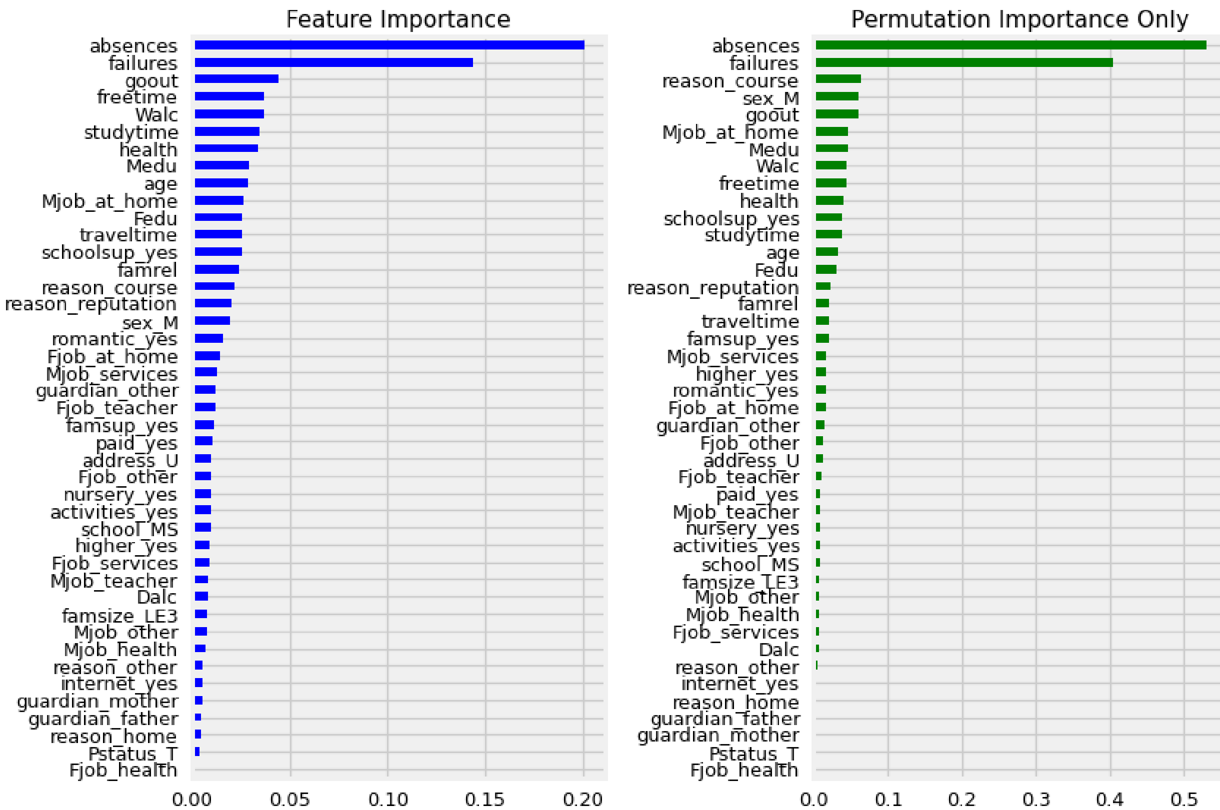


- According to the permutation importance results:
  - Zipcode, sqft_living,and grade were the most important features in predicting price

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Permutation Importance vs Feature Importance

In [16]:
```
fig, axes = plt.subplots(ncols=2,figsize=(12,8))

importances.sort_values().plot(kind='barh',color='blue',ax=axes[0],title='Featu
permutation_importances.sort_values().plot(kind='barh',color='green',ax=axes[1]
fig.tight_layout()
```



- When compared side by side, we can see that the permutation importances and feature importances were similar, but not exactly the same.
- Both importances had the same top 3 most important features, although in a different order.
  - Importantly the feature that the random forest considered to be the most important feature, grade, had a much lower permutation importance when compared to the the most important feature.

Permutation Importance is Model-Agnostic

- Permutation Importance can be calculated using any sklearn model, not just tree-based models.

In [17]:
```
lin_reg = LinearRegression(fit_intercept=True)
lin_reg.fit(X_train_df, y_train)
coeffs_orig = evaluate_regression(lin_reg, X_train_df, y_train, X_test_df,y_tes
coeffs_orig
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
       R^2  RMSE
Train 0.30  3.91
Test  0.04  4.12
```

Out[17]:

```
failures              -1.79
Mjob_teacher          -1.63
romantic_yes          -1.39
schoolsup_yes         -1.08
famsup_yes            -1.07
reason_course         -0.99
Fjob_other            -0.82
goout                 -0.70
Fjob_services         -0.65
guardian_father       -0.65
age                   -0.34
Dalc                  -0.33
Mjob_other            -0.31
Mjob_at_home          -0.30
reason_home           -0.21
health                -0.21
guardian_mother       -0.20
nursery_yes           -0.13
traveltime            -0.13
Pstatus_T             -0.12
Fedu                  -0.06
absences               0.05
activities_yes         0.06
famrel                 0.12
Fjob_at_home           0.17
freetime               0.27
famsize_LE3            0.29
studytime              0.29
Fjob_health            0.29
Walc                   0.36
paid_yes               0.41
reason_reputation      0.45
Mjob_services          0.54
Medu                   0.58
address_U              0.60
school_MS              0.61
reason_other           0.75
internet_yes           0.76
guardian_other         0.85
sex_M                  1.00
Fjob_teacher           1.02
higher_yes             1.38
Mjob_health            1.69
intercept             14.98
Name: Coefficients, dtype: float64
```

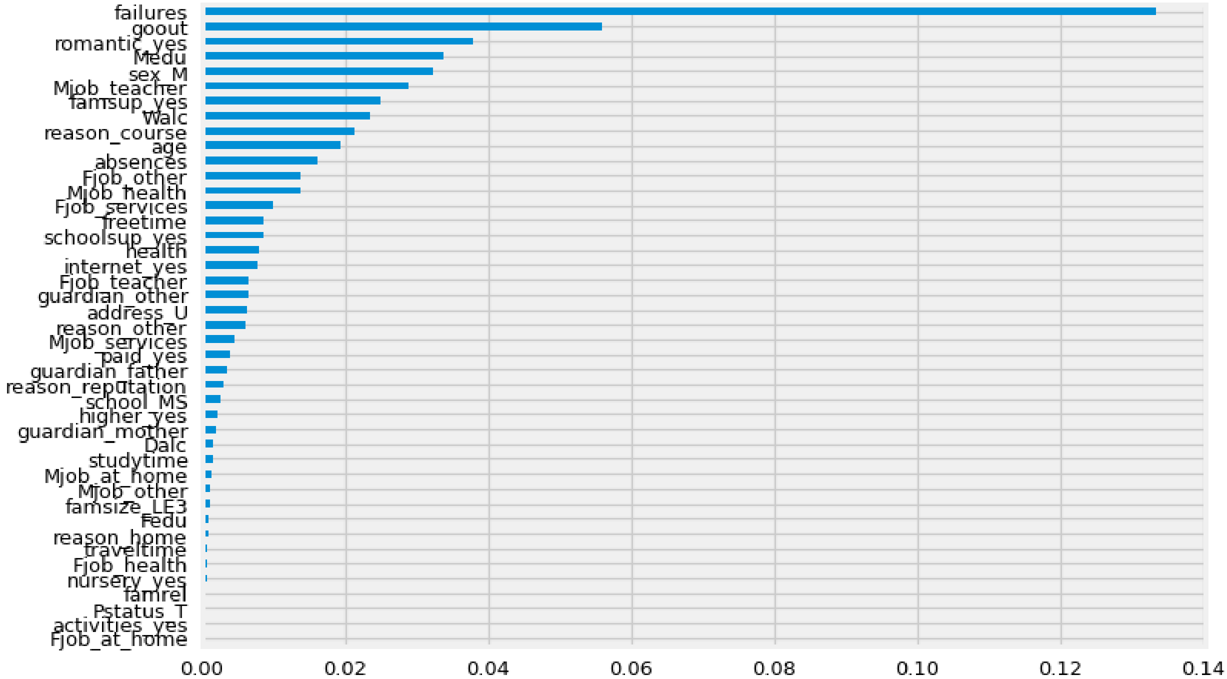
In [18]:

```
## Permutation importance takes a fit mode and test data.

r = permutation_importance(lin_reg, X_train_df, y_train,n_repeats =5)

## can make the mean importances into a series
permutation_importances_linreg = pd.Series(r['importances_mean'],index=X_train_
                           name = 'permutation importance')
permutation_importances_linreg.sort_values().plot(kind='barh')
```

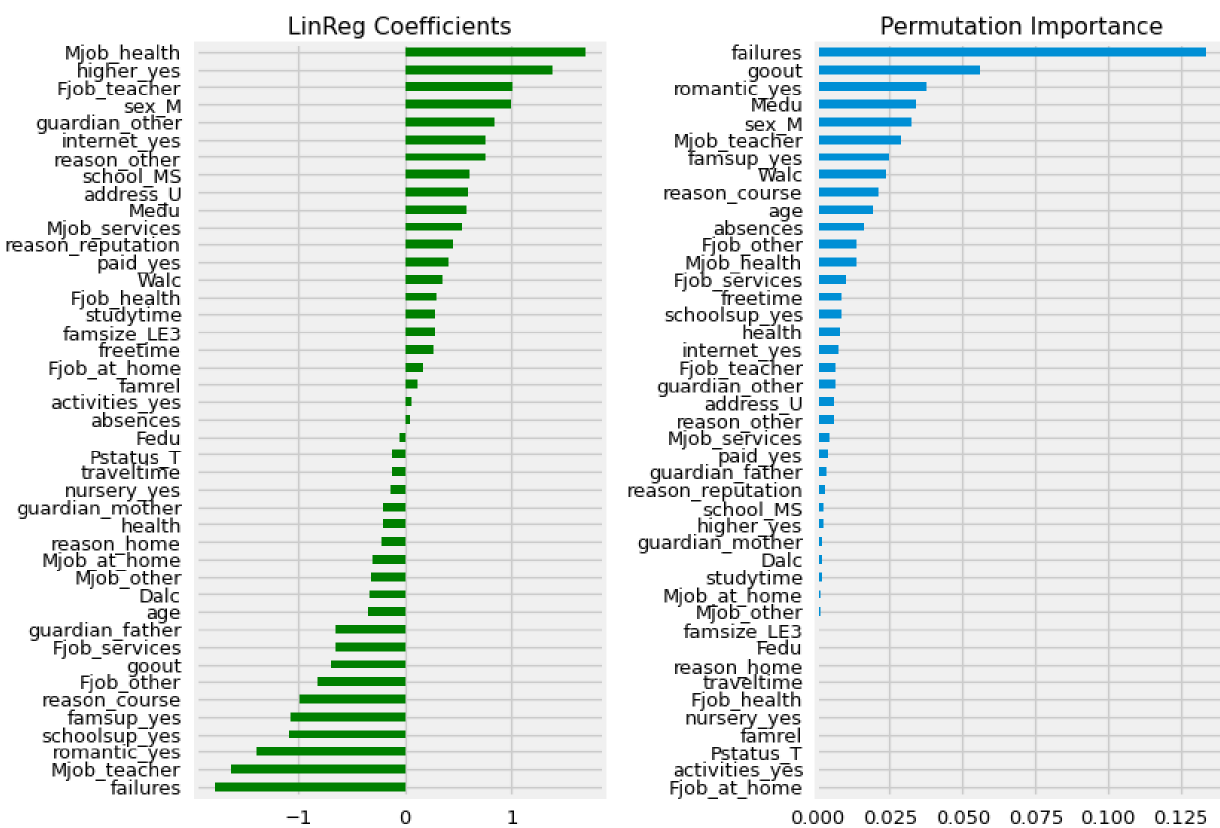
Out[18]:

```
<AxesSubplot:>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [19]:
```
fig, axes = plt.subplots(ncols=2,figsize=(12,8))

coeffs_orig.drop('intercept').sort_values().plot(kind='barh',color='green',ax=a
                                      title='LinReg Coefficients')

permutation_importances_linreg.sort_values().plot(kind='barh',ax=axes[1],
                                                 title='Permutation Importance'

fig.tight_layout()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- Notice how different the sorted coefficients are from the sorted feature importances.
- Since some coefficients are negative, we should try sorting them by their size alone using the absolute value and compare them to the related permutation importance.

In [20]:
```
## Making 1 dataframe combining lin reg coeffs and permutation importance
df_compare = pd.DataFrame({"LinReg Coeffs":coeffs_orig.drop('intercept'),
                           'Permutation Importance':permutation_importances_linr
                           }, index=coeffs_orig.drop('intercept').index)

##  Calculate the rank of the coefficients (according to their abs value)
df_compare['Coeff Rank'] = df_compare['LinReg Coeffs'].abs().rank()

## Sort by rank
df_compare = df_compare.sort_values('Coeff Rank')
df_compare
```

Out[20]:

	LinReg Coeffs	Permutation Importance	Coeff Rank
absences	0.05	0.02	1.00
Fedu	-0.06	0.00	2.00
activities_yes	0.06	0.00	3.00
famrel	0.12	0.00	4.00
Pstatus_T	-0.12	0.00	5.00
traveltime	-0.13	0.00	6.00
nursery_yes	-0.13	0.00	7.00
Fjob_at_home	0.17	0.00	8.00
guardian_mother	-0.20	0.00	9.00
health	-0.21	0.01	10.00
reason_home	-0.21	0.00	11.00
freetime	0.27	0.01	12.00
famsize_LE3	0.29	0.00	13.00
studytime	0.29	0.00	14.00
Fjob_health	0.29	0.00	15.00
Mjob_at_home	-0.30	0.00	16.00
Mjob_other	-0.31	0.00	17.00
Dalc	-0.33	0.00	18.00
age	-0.34	0.02	19.00
Walc	0.36	0.02	20.00
paid_yes	0.41	0.00	21.00
reason_reputation	0.45	0.00	22.00
Mjob_services	0.54	0.00	23.00
Medu	0.58	0.03	24.00

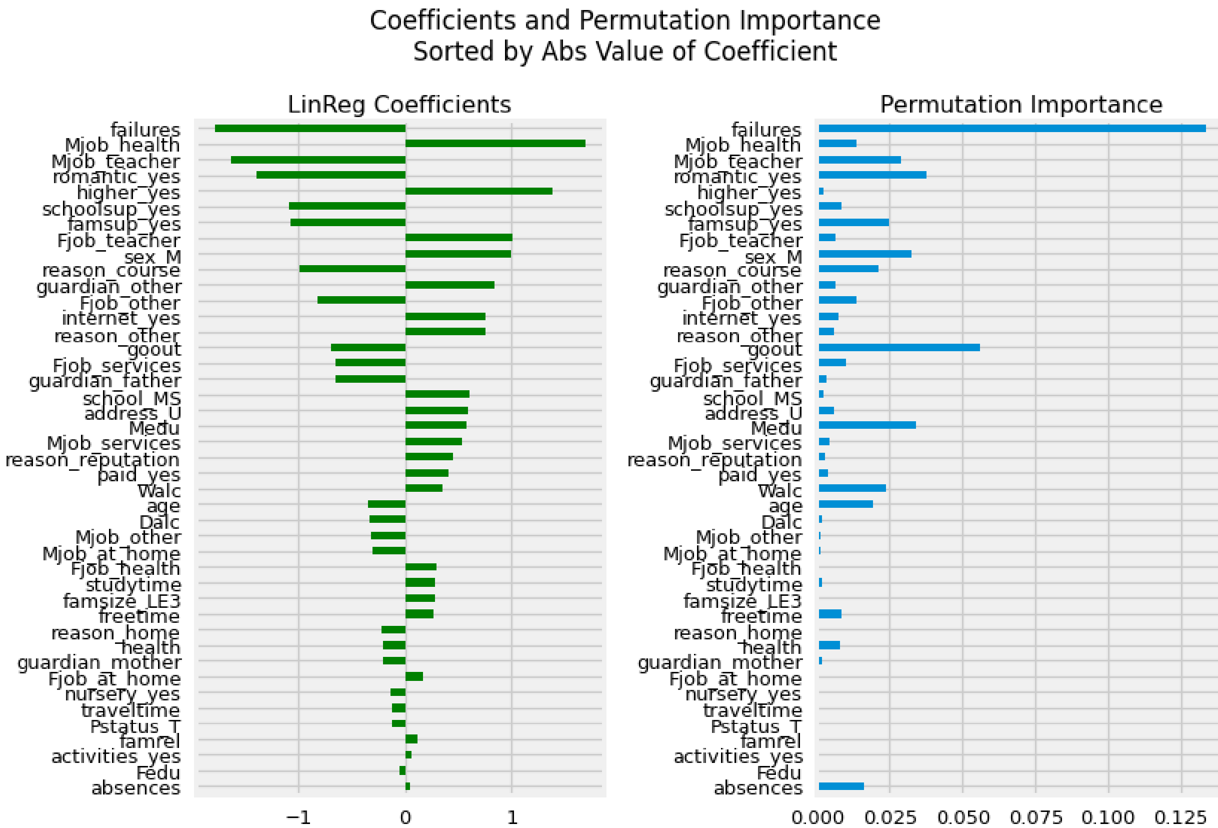
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	LinReg Coeffs	Permutation Importance	Coeff Rank
address_U	0.60	0.01	25.00
school_MS	0.61	0.00	26.00
guardian_father	-0.65	0.00	27.00
Fjob_services	-0.65	0.01	28.00
goout	-0.70	0.06	29.00
reason_other	0.75	0.01	30.00
internet_yes	0.76	0.01	31.00
Fjob_other	-0.82	0.01	32.00
guardian_other	0.85	0.01	33.00
reason_course	-0.99	0.02	34.00
sex_M	1.00	0.03	35.00
Fjob_teacher	1.02	0.01	36.00
famsup_yes	-1.07	0.02	37.00
schoolsup_yes	-1.08	0.01	38.00
higher_yes	1.38	0.00	39.00
romantic_yes	-1.39	0.04	40.00
Mjob_teacher	-1.63	0.03	41.00
Mjob_health	1.69	0.01	42.00
failures	-1.79	0.13	43.00

In [21]:

```
## visually compare the coefficients and importances (same order)
fig, axes = plt.subplots(ncols=2,figsize=(12,8))

df_compare['LinReg Coeffs'].plot(kind='barh',color='green',ax=axes[0],
                                  title='LinReg Coefficients')

df_compare['Permutation Importance'].plot(kind='barh',ax=axes[1],
                                           title='Permutation Importance'
fig.suptitle('Coefficients and Permutation Importance\nSorted by Abs Value of C
fig.tight_layout()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



> `Add example interpretation for stakeholder?`

Advantages/Disadvantages of Permutation Importance

- Advantages:
  - Model agnostic (can be used on any model)
  - Avoids the biases of built-in feature importances.
- Disadvantages:
  - Only positive values (don't know which way each feature pushes the model's predictions)

Beyond sklearn

- Next lesson, we will introduce additional packages designed to explain machine learning models with greater detail.

Summary

- In this lesson we learned about feature importance and its advantages and disadvangates. We also implemented a scikit-learn tool that calculated similar importances, but in a more fair/thoughtful way.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- Next lesson, we will introduce some additional packages whose entire purpose is to better explain how models make their predictions.

APPENDIX

Saving the Model

In [23]:
```
## saving variables for next lesson/notebook
import joblib

export = {'X_train':X_train_df,
         'y_train': y_train,
         'X_test':X_test_df,
         "y_test": y_test,
         'preprocessor':preprocessor,
         'lin_reg':lin_reg,
         'rf_reg':rf_reg}

joblib.dump(export, 'lesson_03.joblib')
```

Out[23]:
```
['lesson_03.joblib']
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js