Regression

September 16, 2019

0.1 Pedram Jahangiry, Fall 2019

1 Regression Analysis:

A linear Regression is a **linear approximation** of a **causal relationship** between two or more variables

1.1 Multiple Regression

First we need to import the libraries:

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

sns.set()  #if you want to use seaborn themes with matplotlib functions
```

1.1.1 Data Preprocessing

[2]:
```
df = pd.read_csv("wage.csv")
df.head()
```

[2]:
```
       wage  hours   IQ  educ  exper  tenure  age  married  black  meduc  \
0  769000.0     40   93    12     11       2   31        1      0    8.0
1  808000.0     50  119    18     11      16   37        1      0   14.0
2  825000.0     40  108    14     11       9   33        1      0   14.0
3  650000.0     40   96    12     13       7   32        1      0   12.0
4  562000.0     40   74    11     14       5   34        1      0    6.0

   feduc
0    8.0
1   14.0
2   14.0
3   12.0
4   11.0
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 935 entries, 0 to 934
Data columns (total 11 columns):
wage       935 non-null float64
hours      935 non-null int64
IQ         935 non-null int64
educ       935 non-null int64
exper      935 non-null int64
tenure     935 non-null int64
age        935 non-null int64
married    935 non-null int64
black      935 non-null int64
meduc      857 non-null float64
feduc      741 non-null float64
dtypes: float64(3), int64(8)
memory usage: 80.4 KB
```

[4]:
```
df.describe().T
```

[4]:
```
         count           mean            std       min       25%       50%  \
wage     935.0  957945.454545  404360.822474  115000.0  669000.0  905000.0
hours    935.0      43.929412       7.224256      20.0      40.0      40.0
IQ       935.0     101.282353      15.052636      50.0      92.0     102.0
educ     935.0      13.468449       2.196654       9.0      12.0      12.0
exper    935.0      11.563636       4.374586       1.0       8.0      11.0
tenure   935.0       7.234225       5.075206       0.0       3.0       7.0
age      935.0      33.080214       3.107803      28.0      30.0      33.0
married  935.0       0.893048       0.309217       0.0       1.0       1.0
black    935.0       0.128342       0.334650       0.0       0.0       0.0
meduc    857.0      10.682614       2.849756       0.0       8.0      12.0
feduc    741.0      10.217274       3.300700       0.0       8.0      10.0

               75%        max
wage     1160000.0  3078000.0
hours         48.0       80.0
IQ           112.0      145.0
educ          16.0       18.0
exper         15.0       23.0
tenure        11.0       22.0
age           36.0       38.0
married        1.0        1.0
black          0.0        1.0
meduc         12.0       18.0
feduc         12.0       18.0
```

[5]:
```
df.isna().sum()
# Alternatively we could use isnull() from pandas.
# pd.isnull(df).sum()
```

[5]:
```
wage          0
hours         0
IQ            0
educ          0
exper         0
tenure        0
age           0
married       0
black         0
meduc        78
feduc       194
dtype: int64
```

1. Because the number of NAs in feduc and meduc is greater that 5% of the observations, we should not keep them in the regression
2. For practice purposes, I will keep the meduc in the features.

[6]:
```
df.drop('feduc', axis=1, inplace=True) #why do we need inplace?
```

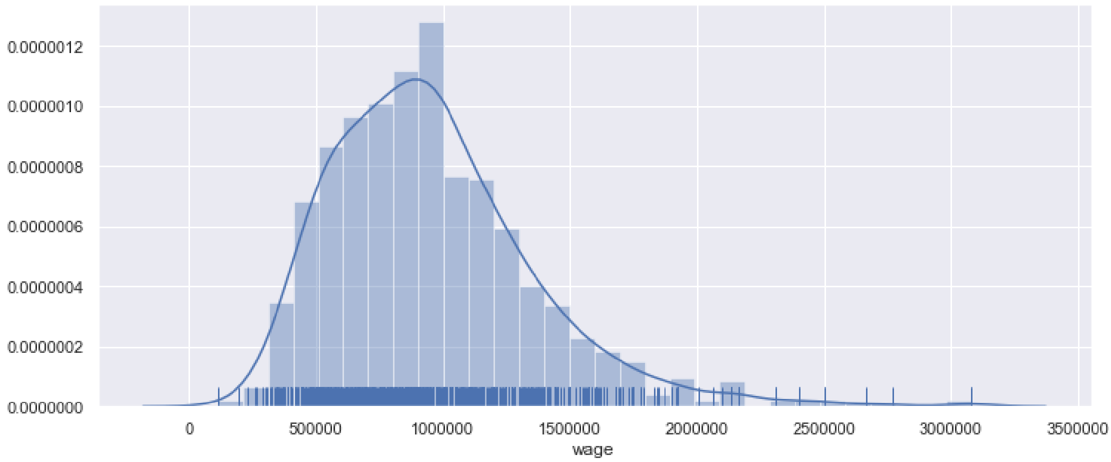
[7]:
```
# we will replace the missing meduc with median. Because the data is left␣
↪skewed
# mean is not a good representation of the central tendency measure.

df['meduc'].fillna(df['meduc'].median(),axis=0, inplace=True )
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 935 entries, 0 to 934
Data columns (total 10 columns):
wage       935 non-null float64
hours      935 non-null int64
IQ         935 non-null int64
educ       935 non-null int64
exper      935 non-null int64
tenure     935 non-null int64
age        935 non-null int64
married    935 non-null int64
black      935 non-null int64
meduc      935 non-null float64
dtypes: float64(2), int64(8)
memory usage: 73.1 KB
```

1.1.2 Data visualization

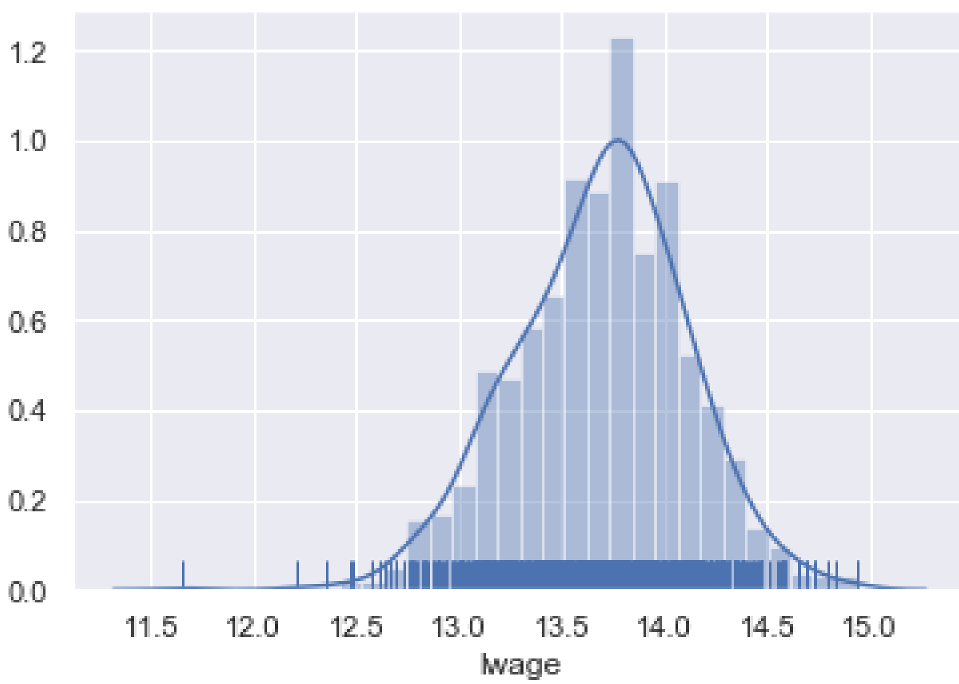
[8]:
```
plt.figure(figsize=(12,5))
sns.distplot(df['wage'], bins=30 , rug=True)
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26f9d9bcc18>
```



[9]:
```
# Need to do log transformation to avoid potential heteroskedasticity
df['lwage']= np.log(df['wage'])
sns.distplot(df['lwage'], bins=30 , rug=True)
```

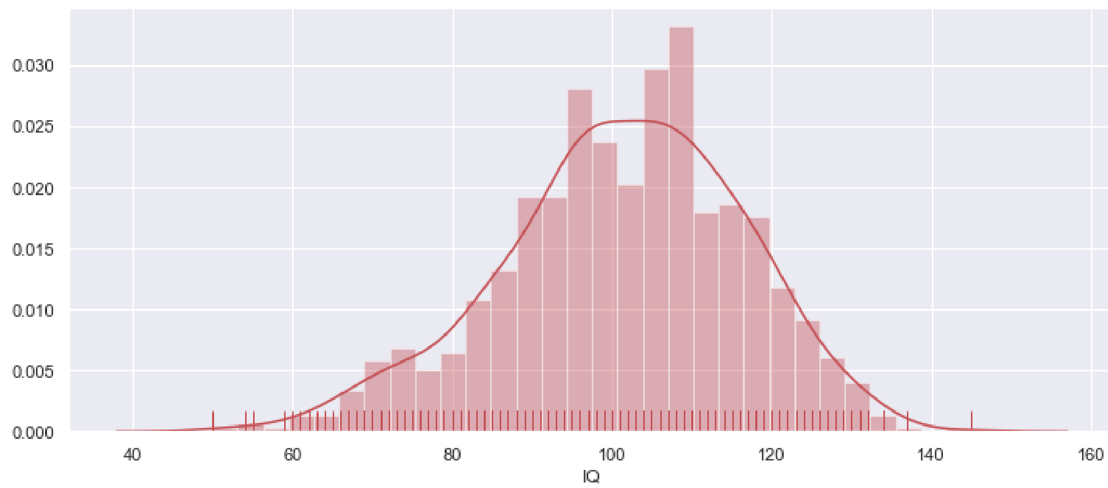
[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x26f9dd36a90>
```



[10]:
```
df.drop('wage', axis=1, inplace=True)
```

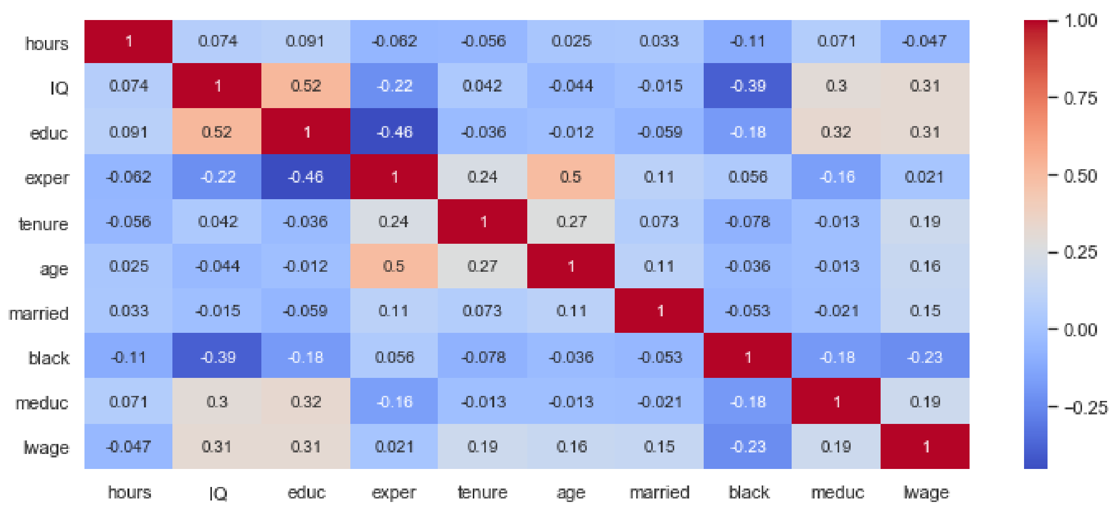
[11]:
```
plt.figure(figsize=(12,5))
sns.distplot(df['IQ'], bins=30 ,color='r' , rug=True)
```

[11]: <matplotlib.axes._subplots.AxesSubplot at 0x26f9de1a780>



[12]:
```
plt.figure(figsize=(12,5))
sns.heatmap(df.corr(), cmap='coolwarm',annot=True)
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x26f9e132a58>



[13]:
```
sns.pairplot(df[['lwage', 'IQ', 'educ','exper','tenure','age']])
```

[13]: <seaborn.axisgrid.PairGrid at 0x26f9e143940>

1.1.3 Dealing with dummy variables

[14]:
```
df2 = df.copy()  # make sure you put df.copy(). Why?
df2['status']=df2['married'].map({1:'Married', 0:'Single'})
df2.drop('married',axis=1 ,inplace=True)
df2.head()
```

[14]:

	hours	IQ	educ	exper	tenure	age	black	meduc	lwage	status
0	40	93	12	11	2	31	0	8.0	13.552846	Married
1	50	119	18	11	16	37	0	14.0	13.602317	Married
2	40	108	14	11	9	33	0	14.0	13.623139	Married

```
3     40   96    12      13         7   32         0    12.0  13.384728  Married
4     40   74    11      14         5   34         0     6.0  13.239257  Married
```

[15]:
```
df2 = pd.get_dummies(df2, drop_first=True)
df2.head()
```

[15]:
```
   hours   IQ  educ  exper  tenure  age  black  meduc      lwage  \
0     40   93    12     11       2   31      0    8.0  13.552846
1     50  119    18     11      16   37      0   14.0  13.602317
2     40  108    14     11       9   33      0   14.0  13.623139
3     40   96    12     13       7   32      0   12.0  13.384728
4     40   74    11     14       5   34      0    6.0  13.239257

   status_Single
0              0
1              0
2              0
3              0
4              0
```

in this example, since black and married are already dummy variables, we don't need to do any thing else.

1.1.4 Defining the variables and splitting the data

[16]:
```
y = df['lwage']
X = df.drop('lwage', axis=1) # becareful inplace= False

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=100)

len(X_train)/len(X)
```

[16]:
```
0.8
```

1.2 Linear Regression with StatsModels

Statsmodel is great for learning the theory of regression models. Also Statsmodel works perfectly with pandas dataframe. However, sklearn is a more **practical** package preferred by ML practicitionairs to apply regression analysis.

[17]:
```
# Add a constant
X_test_wc = sm.add_constant(X_test)
X_train_wc = sm.add_constant(X_train)
```

```
C:\Users\Pedram\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389:
FutureWarning: Method .ptp is deprecated and will be removed in a future
version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

[18]:
```
X_train_wc.head()
```

[18]:

	const	hours	IQ	educ	exper	tenure	age	married	black	meduc
457	1.0	50	130	14	15	1	33	1	0	12.0
807	1.0	40	104	12	10	2	29	1	0	12.0
859	1.0	60	105	16	12	1	35	1	0	14.0
174	1.0	60	116	12	9	7	30	1	0	12.0
417	1.0	58	113	16	9	0	30	1	0	12.0

[19]:
```
# Fit the model
model = sm.OLS(y_train,X_train_wc)
statsmodels_reg= model.fit()
```

[20]:
```
statsmodels_reg.summary()
```

[20]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  lwage   R-squared:                       0.228
Model:                            OLS   Adj. R-squared:                  0.218
Method:                 Least Squares   F-statistic:                     24.19
Date:                Mon, 16 Sep 2019   Prob (F-statistic):           1.82e-36
Time:                        18:18:27   Log-Likelihood:                -296.16
No. Observations:                 748   AIC:                             612.3
Df Residuals:                     738   BIC:                             658.5
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         12.0552      0.200     60.323      0.000      11.663      12.448
hours         -0.0051      0.002     -2.771      0.006      -0.009      -0.001
IQ             0.0038      0.001      3.460      0.001       0.002       0.006
educ           0.0450      0.008      5.586      0.000       0.029       0.061
exper          0.0071      0.004      1.702      0.089      -0.001       0.015
tenure         0.0088      0.003      3.198      0.001       0.003       0.014
age            0.0146      0.005      2.753      0.006       0.004       0.025
married        0.1633      0.043      3.773      0.000       0.078       0.248
black         -0.1707      0.043     -3.983      0.000      -0.255      -0.087
meduc          0.0109      0.005      2.085      0.037       0.001       0.021
==============================================================================
Omnibus:                       13.666   Durbin-Watson:                   2.227
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               18.162
Skew:                          -0.197   Prob(JB):                     0.000114
Kurtosis:                       3.654   Cond. No.                     1.79e+03
==============================================================================

Warnings:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.79e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

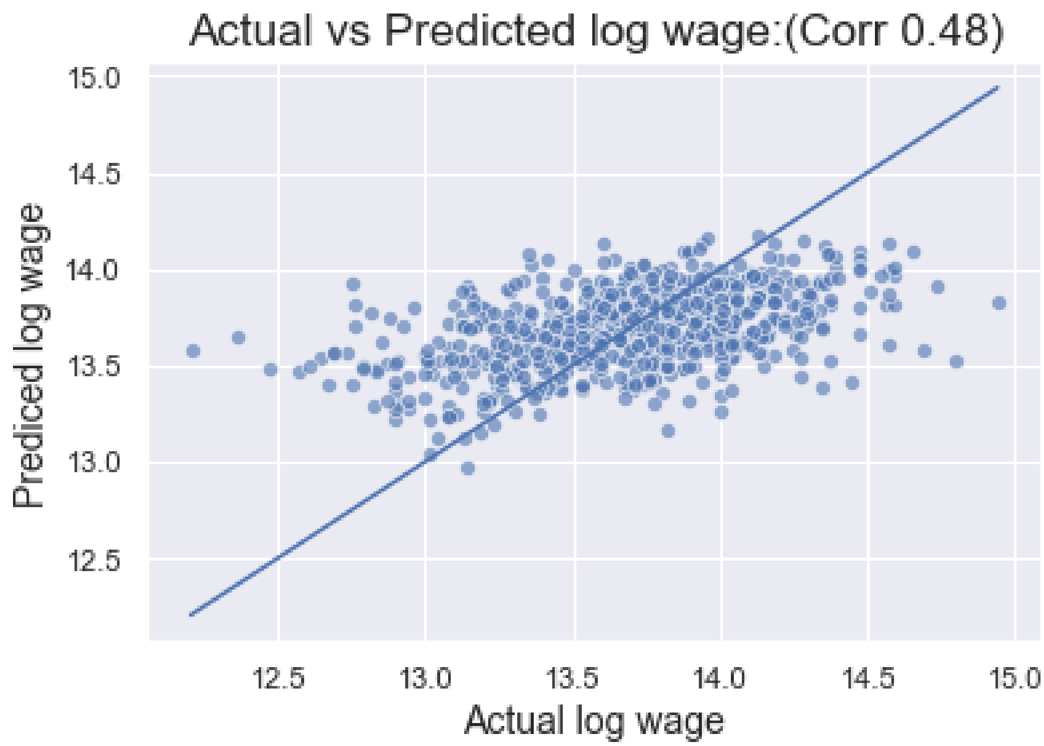
Interpreting the results

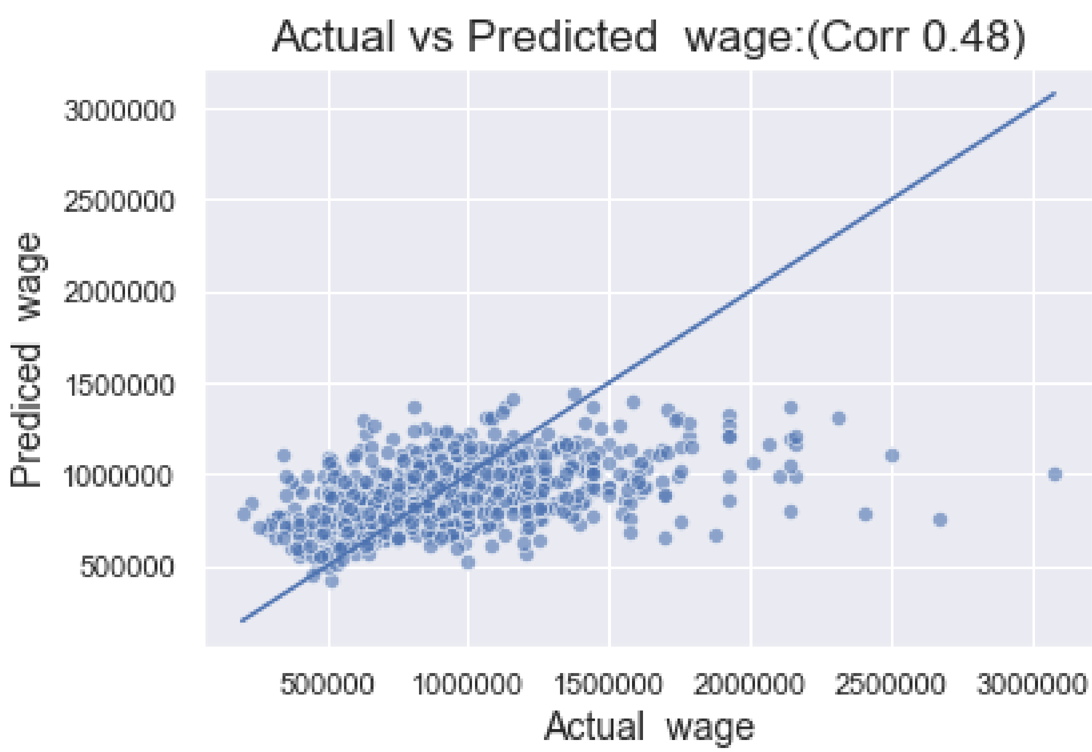
[21]:
```
# try
# statsmodels_reg.
statsmodels_reg.conf_int(alpha=0.01) # if you want to be more conservative set␣
 ↪alpha=0.01
```

[21]:
```
                0          1
const    11.539143  12.571348
hours    -0.009828  -0.000345
IQ        0.000962   0.006618
educ      0.024192   0.065791
exper    -0.003675   0.017885
tenure    0.001694   0.015917
age       0.000903   0.028227
married   0.051533   0.275097
black    -0.281414  -0.060039
meduc    -0.002603   0.024446
```

Graph of Actual vs. Predicted values

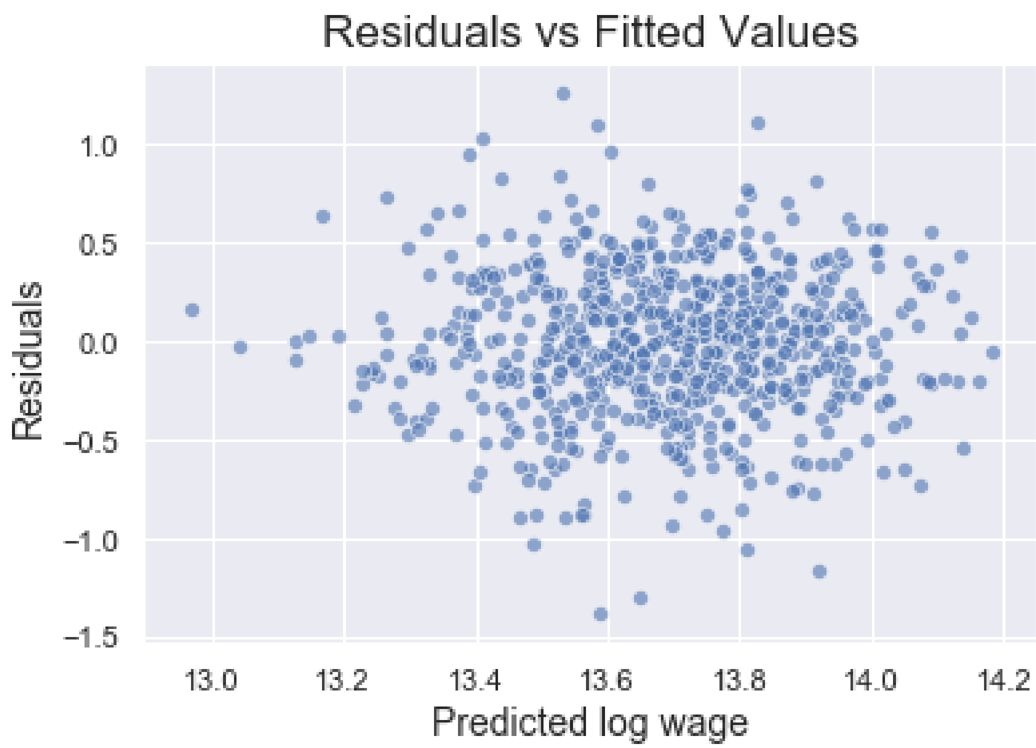
[22]:
```
corr = round(y_train.corr(statsmodels_reg.fittedvalues), 2)
sns.scatterplot(x=y_train, y=statsmodels_reg.fittedvalues, alpha=0.6)
sns.lineplot(y_train, y_train)

plt.xlabel('Actual log wage', fontsize=14)
plt.ylabel('Prediced log wage', fontsize=14)
plt.title(f'Actual vs Predicted log wage:(Corr {corr})', fontsize=17)

plt.show()

sns.scatterplot(x=np.e**y_train, y=np.e**statsmodels_reg.fittedvalues,  alpha=0.
 ↪6)
sns.lineplot(np.e**y_train, np.e**y_train)

plt.xlabel('Actual  wage', fontsize=14)
plt.ylabel('Prediced  wage', fontsize=14)
plt.title(f'Actual vs Predicted  wage:(Corr {corr})', fontsize=17)

plt.show()
```


Actual vs Predicted log wage:(Corr 0.48)
15.0
14.5
14.0
13.5
13.0
12.5
Prediced log wage
12.5
13.0
13.5
14.0
14.5
15.0
Actual log wage

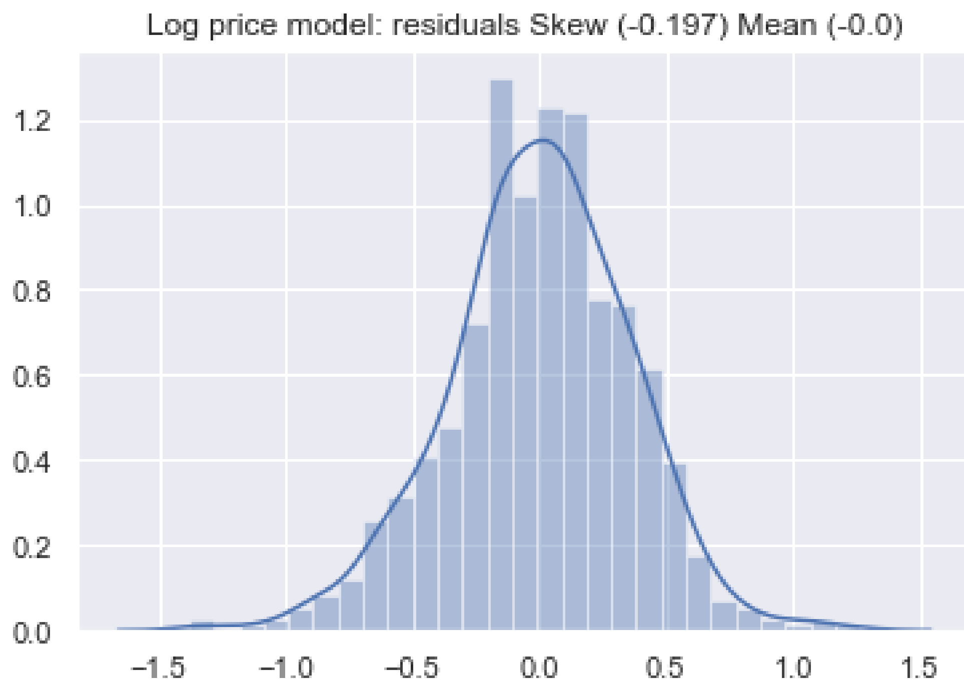

Actual vs Predicted wage:(Corr 0.48)
3000000
2500000
2000000
1500000
1000000
500000
Prediced wage
500000
1000000
1500000
2000000
2500000
3000000
Actual wage

Residuals vs Predicted values

[23]:
```
sns.scatterplot(x=statsmodels_reg.fittedvalues, y=statsmodels_reg.resid ,␣
 ↪alpha=0.6)

plt.xlabel('Predicted log wage', fontsize=14)
plt.ylabel('Residuals', fontsize=14)
plt.title('Residuals vs Fitted Values', fontsize=17)

plt.show()
```



[24]:
```
resid_mean = round(statsmodels_reg.resid.mean(), 3)
resid_skew = round(statsmodels_reg.resid.skew(), 3)

sns.distplot(statsmodels_reg.resid)
plt.title(f'Log price model: residuals Skew ({resid_skew}) Mean ({resid_mean})')
plt.show()
```



[25]:
```
# Mean Squared Error
MSE = round(statsmodels_reg.mse_resid, 3)
MSE
```

[25]:
```
0.131
```

1.3 Linear Regression with Scikit-Learn

So far we have only worked with data frames using Pandas. Now we may need to transform our data into arrays by using numpy because sklearn uses arrays instead of data frames.

Scikit-learn is a very powerful package enabling you to do almost everything in machine learning including Regression, Classification, Clustering, SVM and Dimensionality reduction. However, I don't recommend sklearn for deep learning algorithms. Pytorch, Tensorflow and Keras are better alternatives for deep learning.

Note: with sklearn, we don't need to add constants mannually.

[26]:
```
from sklearn.linear_model import LinearRegression
```

[27]:
```
sklearn_reg = LinearRegression()
```

[28]:
```
sklearn_reg.fit(X_train, y_train)
```

[28]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

1.3.1 Generating results

try reg.

[29]:
```
# The coefficients of the regression
sklearn_reg.coef_
```

[29]:
```
array([-0.00508645,  0.00379004,  0.04499157,  0.00710532,  0.00880574,
        0.01456483,  0.16331493, -0.17072619,  0.01092151])
```

[30]:
```
# The intercept of the regression
sklearn_reg.intercept_
```

[30]:
```
12.055245334454883
```

[31]:
```
# The R-squared of the regression
sklearn_reg.score(X_train,y_train)
```

[31]:
```
0.2278261765190378
```

[32]:
```
X_train.head()
```

[32]:

	hours	IQ	educ	exper	tenure	age	married	black	meduc
457	50	130	14	15	1	33	1	0	12.0
807	40	104	12	10	2	29	1	0	12.0
859	60	105	16	12	1	35	1	0	14.0
174	60	116	12	9	7	30	1	0	12.0
417	58	113	16	9	0	30	1	0	12.0

[33]:
```
# If we want to find the Adjusted R-squared we can do so by knowing the R2, the
↪# observations, the # features
R2 = sklearn_reg.score(X_train,y_train)
n = X_train.shape[0]
p = X_train.shape[1]

# We find the Adjusted R-squared using the formula
adjusted_R2 = 1-(1-R2)*(n-1)/(n-p-1)
adjusted_R2
```

[33]:
```
0.21840942257414797
```

[34]:
```
# Let's create a new data frame with the names of the features
reg_summary = pd.DataFrame(data = X_train.columns.values, columns=['Features'])
reg_summary ['Coefficients'] = sklearn_reg.coef_
reg_summary
```

[34]:

	Features	Coefficients
0	hours	-0.005086
1	IQ	0.003790
2	educ	0.044992
3	exper	0.007105
4	tenure	0.008806
5	age	0.014565
6	married	0.163315

```
7      black      -0.170726
8      meduc       0.010922
```

1.3.2 Further Diagnostic tests

Multicollinearity sklearn does not have a built-in way to check for multicollinearity. The main reasons is that this is an issue well covered in statistical frameworks and not in ML ones. However, we can use statsmodels to run the VIF test.

[35]:
```
X_train.columns.values
```

[35]:
```
array(['hours', 'IQ', 'educ', 'exper', 'tenure', 'age', 'married',
       'black', 'meduc'], dtype=object)
```

[36]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[37]:
```
collinearity = X_train[['hours','IQ','educ','exper','tenure','age', 'married',␣
 ↪'black', 'meduc']]
VIF = pd.DataFrame()
```

[38]:
```
VIF["Features"] = collinearity.columns
VIF["VIF"] = [variance_inflation_factor(collinearity.values, i) for i in␣
 ↪range(collinearity.shape[1])]

VIF
```

[38]:

	Features	VIF
0	hours	33.859663
1	IQ	60.857443
2	educ	68.139961
3	exper	15.038459
4	tenure	3.456928
5	age	116.108410
6	married	9.420645
7	black	1.273572
8	meduc	19.384284

heteroskedasticity

[39]:
```
# using the White test for example:
from statsmodels.stats.diagnostic import het_white
```

[40]:
```
hetero = X_train_wc
F_stat_pvalue = het_white(statsmodels_reg.resid.values, hetero.values,␣
 ↪retres=False)[3]
F_stat_pvalue
```

[40]:
```
0.0014769435236610107
```

1.3.3 Testing

Once we have trained and fine-tuned our model, we can proceed to testing it. Testing is done on a dataset that the algorithm has never seen

[41]:
```
# Using our statmodels results.
statsmodels_reg.predict(X_test_wc).head(4)
```

[41]:
```
143    13.725665
229    13.753259
116    13.664840
134    13.565368
dtype: float64
```

[42]:
```
# Using our sklearn reg.
sklearn_reg.predict(X_test)[0:4]
```

[42]:
```
array([13.72566539, 13.75325897, 13.66484037, 13.56536839])
```

[43]:
```
y_hat_test = sklearn_reg.predict(X_test)
log_predictions = pd.DataFrame( {'Actuals':y_test , 'Predictions': y_hat_test})
predictions = np.exp(log_predictions)
predictions.tail()

# You can reset the index if you wish. How?
```

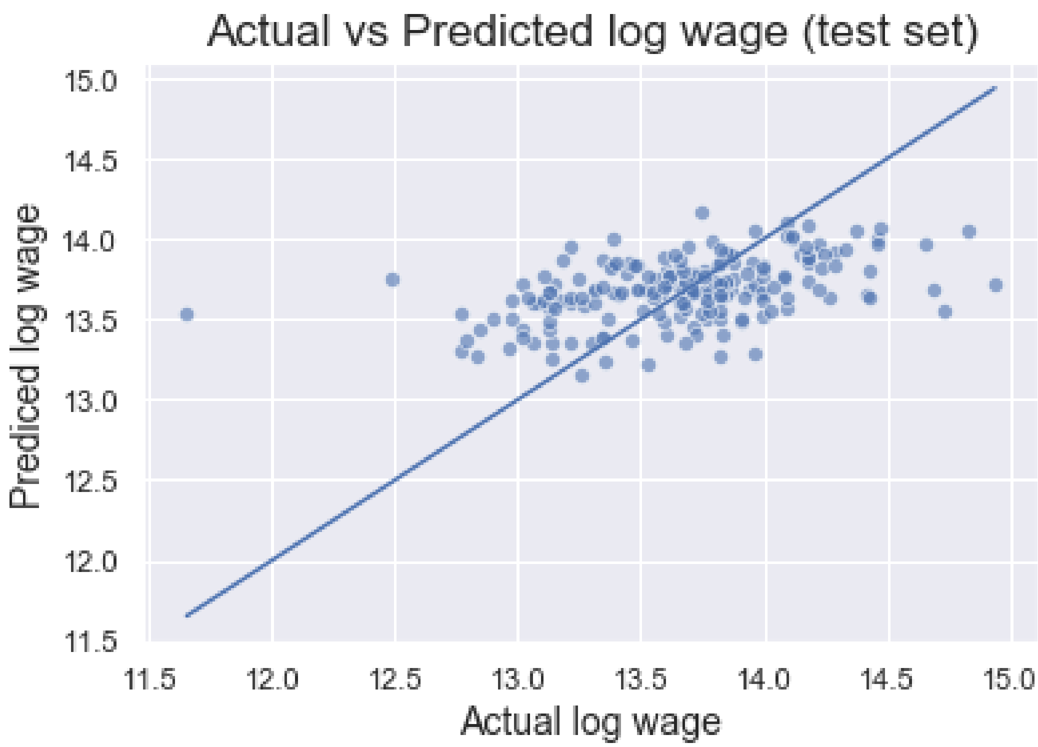
[43]:
```
       Actuals   Predictions
471   511000.0  7.827895e+05
191   840000.0  1.093139e+06
688   865000.0  8.161840e+05
10    930000.0  1.435088e+06
420  1850000.0  8.345331e+05
```

[44]:
```
# Additionally, we can calculate the difference and percentage difference␣
↪between the targets and the predictions
predictions['Residuals'] = predictions['Predictions'] - predictions['Actuals']
predictions['Difference%'] = np.absolute(predictions['Residuals']/
↪predictions['Actuals']*100)
predictions.round().tail(5)
```

[44]:
```
       Actuals  Predictions  Residuals  Difference%
471   511000.0     782789.0   271789.0         53.0
191   840000.0    1093139.0   253139.0         30.0
688   865000.0     816184.0   -48816.0          6.0
10    930000.0    1435088.0   505088.0         54.0
420  1850000.0     834533.0 -1015467.0         55.0
```

[45]:
```
sns.scatterplot(x=y_test, y=y_hat_test, alpha=0.6)
sns.lineplot(y_test, y_test)

plt.xlabel('Actual log wage', fontsize=14)
plt.ylabel('Prediced log wage', fontsize=14)
```

```
plt.title('Actual vs Predicted log wage (test set)', fontsize=17)
plt.show()
```



[46]:
```
sns.scatterplot(x=predictions['Predictions'], y=predictions['Residuals'] ,␣
 ↪alpha=0.6)
plt.show()

# try to replicate this residual plot using log wages
```

2 Plotting the Simple Regression Line

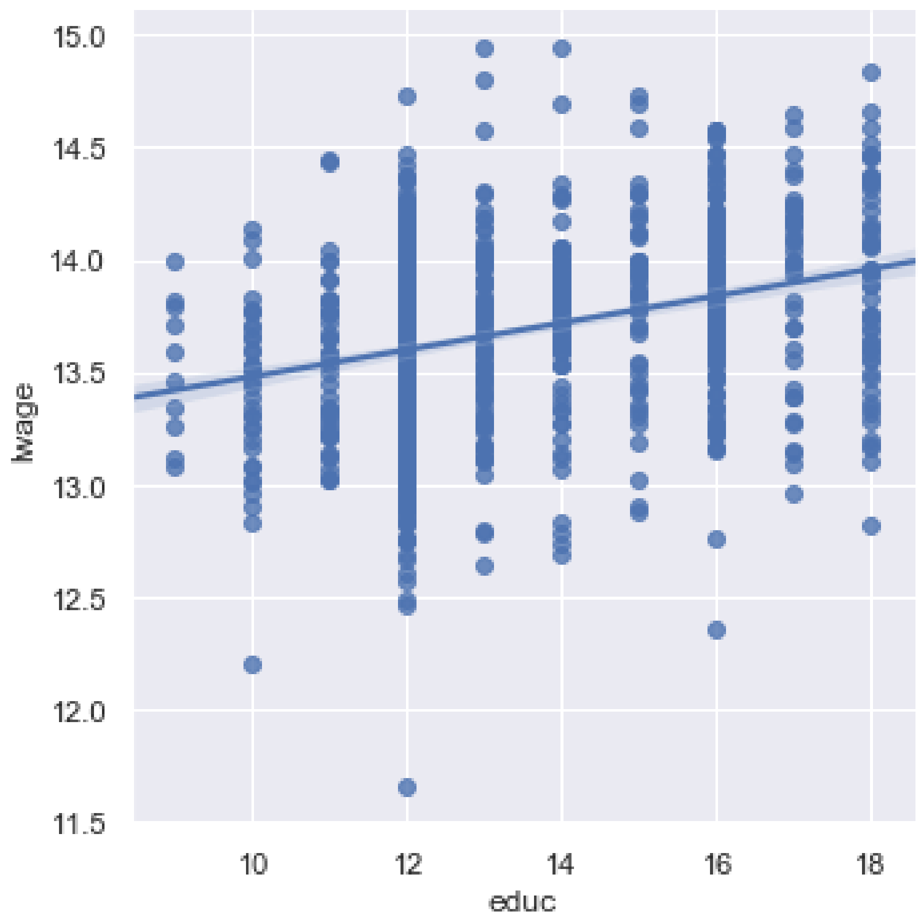
Let's pick one of the most significant features and make a simple regression model with that.

[47]:
```
df.head(4)
```

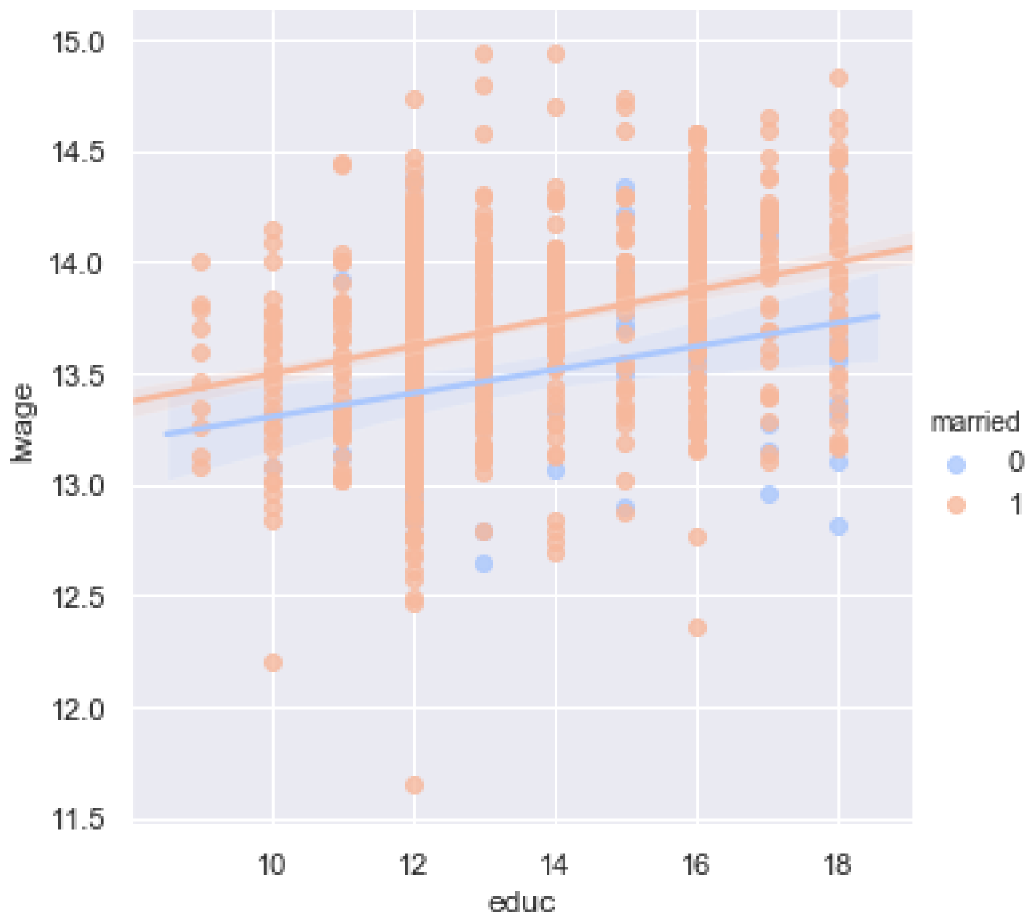
[47]:

	hours	IQ	educ	exper	tenure	age	married	black	meduc	lwage
0	40	93	12	11	2	31	1	0	8.0	13.552846
1	50	119	18	11	16	37	1	0	14.0	13.602317
2	40	108	14	11	9	33	1	0	14.0	13.623139
3	40	96	12	13	7	32	1	0	12.0	13.384728

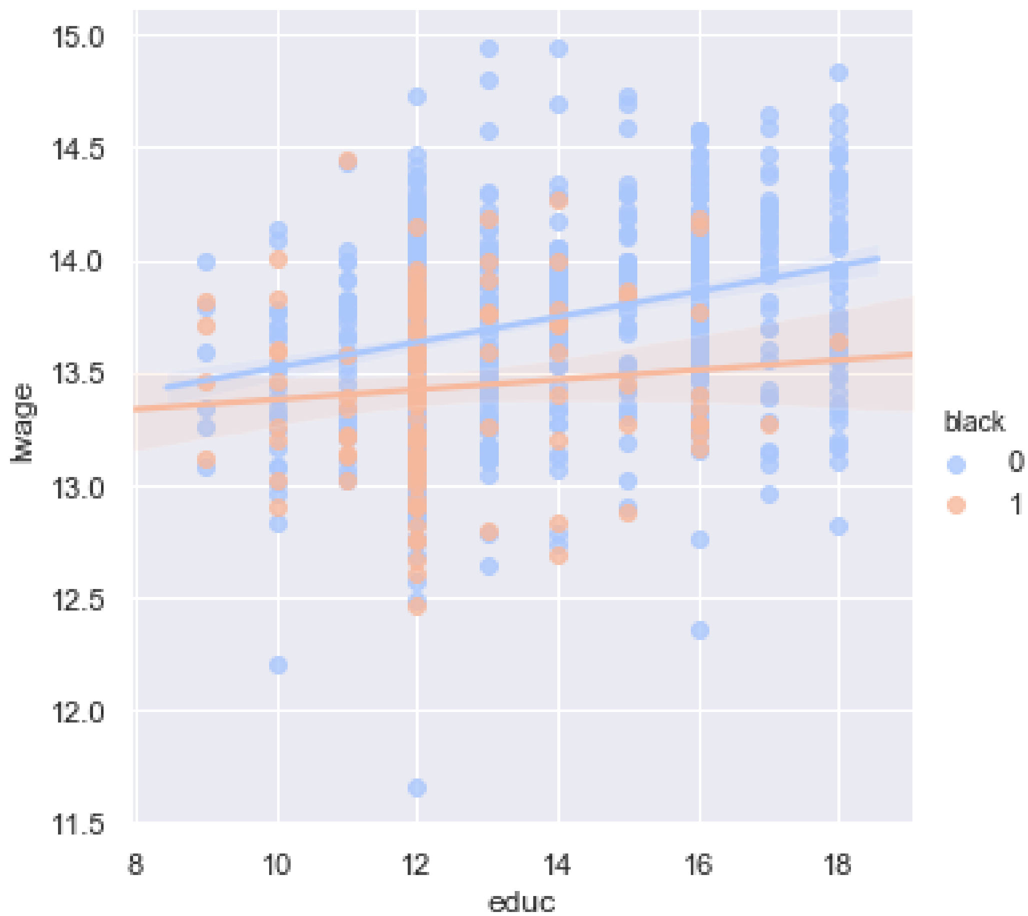
[48]:
```
sns.lmplot(x='educ',y='lwage',data=df)
plt.show()
```



[49]:
```
sns.lmplot(x='educ',y='lwage',data=df, hue='married',palette='coolwarm') # how␣
↪do you interpret this one?
plt.show()
```



[50]:
```
sns.lmplot(x='educ',y='lwage',data=df, hue='black',palette='coolwarm')  # how␣
↪do you interpret this one?
plt.show()
```



[51]:
```
sns.lmplot(x='educ',y='lwage',data=df, hue='black',palette='coolwarm',␣
 ↪col='married')  # how do you interpret this one?
plt.show()
```