In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from itertools import combinations
import tensorflow as tf
%matplotlib inline
```

```
ype is deprecated; in a future version of numpy, it will be understood as (ty
pe, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

The data set used is "Campus Recruitment Academic and Employability Factors influencing placement" by Ben Roshan D [link (https://www.kaggle.com/benroshan/factors-affecting-campus-placement)](https://www.kaggle.com/benroshan/factors-affecting-campus-placement)

The best description is the one given my Roshan itself: "This data set consists of Placement data of students in our campus. It includes secondary and higher secondary school percentage and specialization. It also includes degree specialization, type and Work experience and salary offers to the placed students"(Roshan). It has 15 different features: sl_no, gender, ssc_p, ssc_b, hsc_p, hsc_b, hsc_s, degree_p, degree_t, workex, etest_p, specialisation, mba_p, status, and salary.

- sl_no: Serial Number of index
- gender: Gender (M: male, F: female)
- ssc_p: Secondary Education percentage- 10th Grade
- ssc_b: Board of Education (Central,Others)
- hsc_p: Higher Secondary Education percentage- 12th Grade
- hsc_b: Board of Education (Central, Others)
- hsc_s: Specialization in Higher Secondary Education
- degree_p: Degree Percentage
- degree_t: Under Graduation(Degree type)- Field of degree education
- workex: Work Experience (Yes, No)
- etest_p:
- specialisation: type of specialisation
- mba_p: mba percentage
- status: if working or not (Not Placed, Placed)
- salary: salary of the person

In [2]:
```
df=pd.read_csv('datasets_596958_1073629_Placement_Data_Full_Class.csv')
df
```

Out[2]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No
...	...	...	...	...	...	...	...	...	...	...
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	No
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	No
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes
213	214	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	No
214	215	M	62.00	Central	58.00	Others	Science	53.00	Comm&Mgmt	No

215 rows × 15 columns

EDA

In [3]:
```
df[['ssc_p','hsc_p','degree_p','etest_p','mba_p','salary']].describe()
```

Out[3]:

	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
count	215.000000	215.000000	215.000000	215.000000	215.000000	148.000000
mean	67.303395	66.333163	66.370186	72.100558	62.278186	288655.405405
std	10.827205	10.897509	7.358743	13.275956	5.833385	93457.452420
min	40.890000	37.000000	50.000000	50.000000	51.210000	200000.000000
25%	60.600000	60.900000	61.000000	60.000000	57.945000	240000.000000
50%	67.000000	65.000000	66.000000	71.000000	62.000000	265000.000000
75%	75.700000	73.000000	72.000000	83.500000	66.255000	300000.000000
max	89.400000	97.700000	91.000000	98.000000	77.890000	940000.000000

Here we can see a basic statiscal analysis of the data, the feature of interes in this notebook is the salary

In [4]:
```
df.corr()
```

Out[4]:

	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
sl_no	1.000000	-0.078155	-0.085711	-0.088281	0.063636	0.022327	0.063764
ssc_p	-0.078155	1.000000	0.511472	0.538404	0.261993	0.388478	0.035330
hsc_p	-0.085711	0.511472	1.000000	0.434206	0.245113	0.354823	0.076819
degree_p	-0.088281	0.538404	0.434206	1.000000	0.224470	0.402364	-0.019272
etest_p	0.063636	0.261993	0.245113	0.224470	1.000000	0.218055	0.178307
mba_p	0.022327	0.388478	0.354823	0.402364	0.218055	1.000000	0.175013
salary	0.063764	0.035330	0.076819	-0.019272	0.178307	0.175013	1.000000

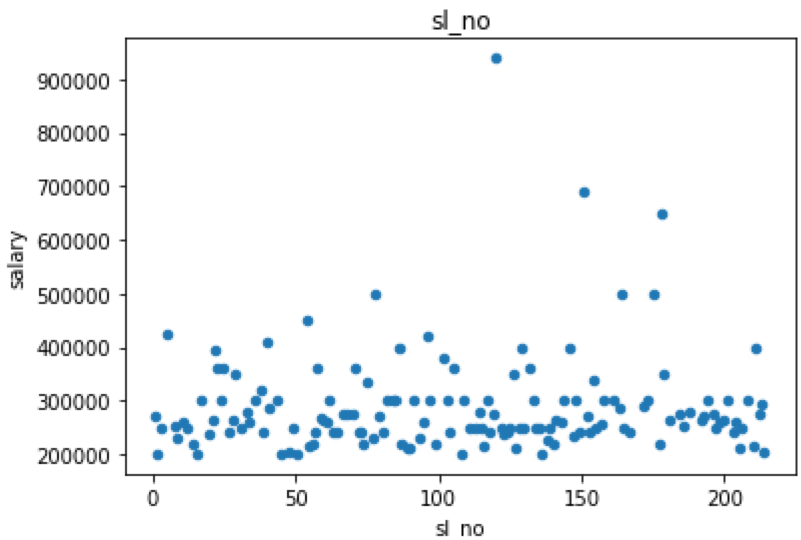
Here are the variable correlations

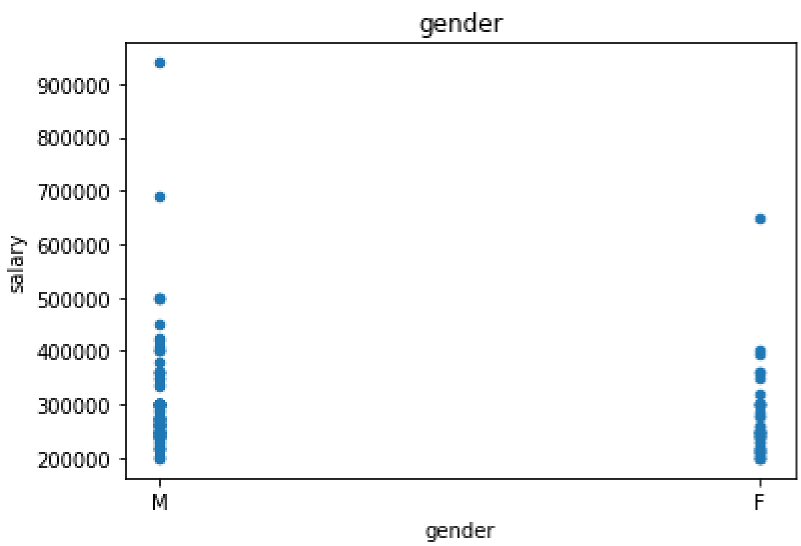
In [5]:
```
df.corr().loc['salary']
```

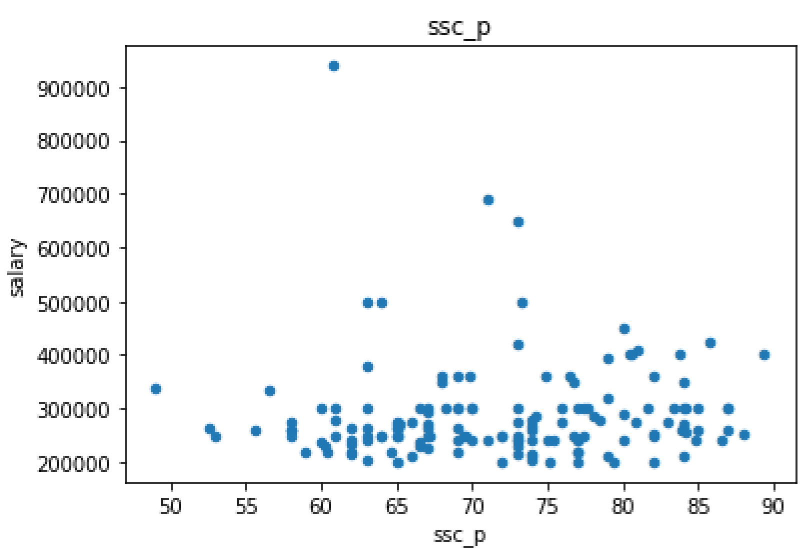
Out[5]:
```
sl_no        0.063764
ssc_p        0.035330
hsc_p        0.076819
degree_p    -0.019272
etest_p      0.178307
mba_p        0.175013
salary       1.000000
Name: salary, dtype: float64
```

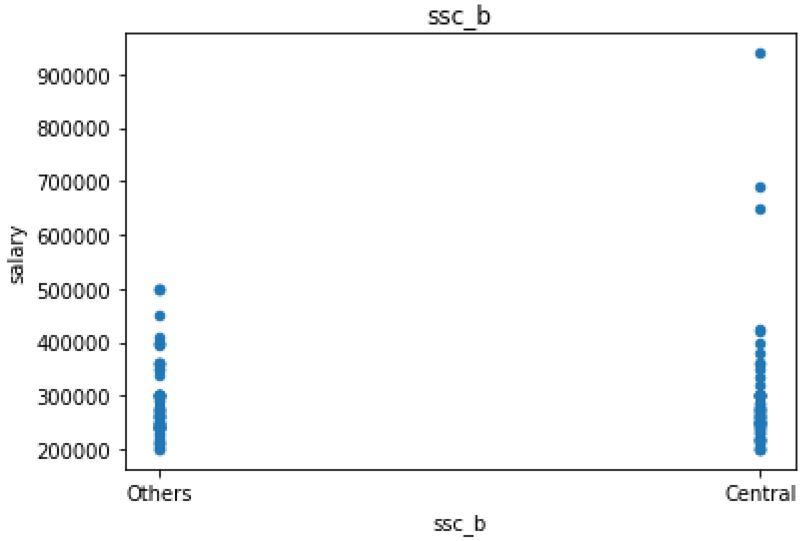
The correlations of our variable of interest (salary) There is yet no correlation for degree type since it is a categorical string feature

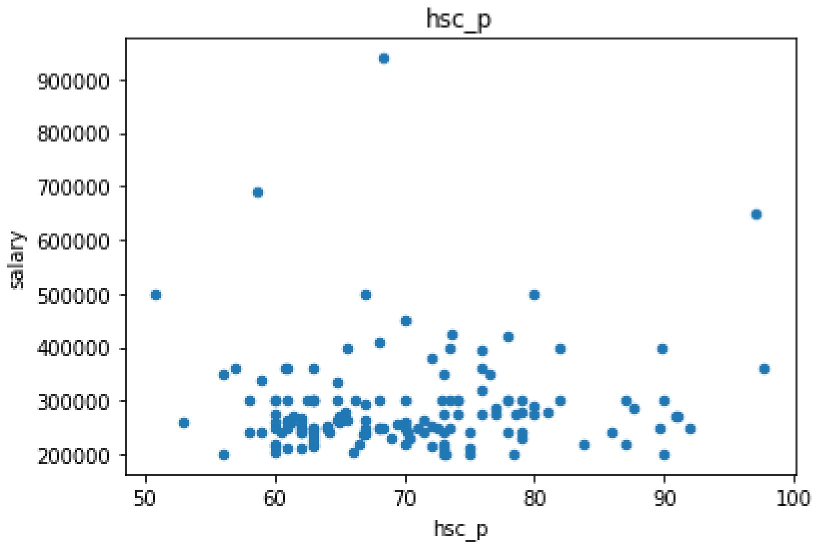
In [6]:
```
for c in df.columns:
    try:
        df.plot(c,'salary',kind='scatter',title=c)
    except:
        pass
```

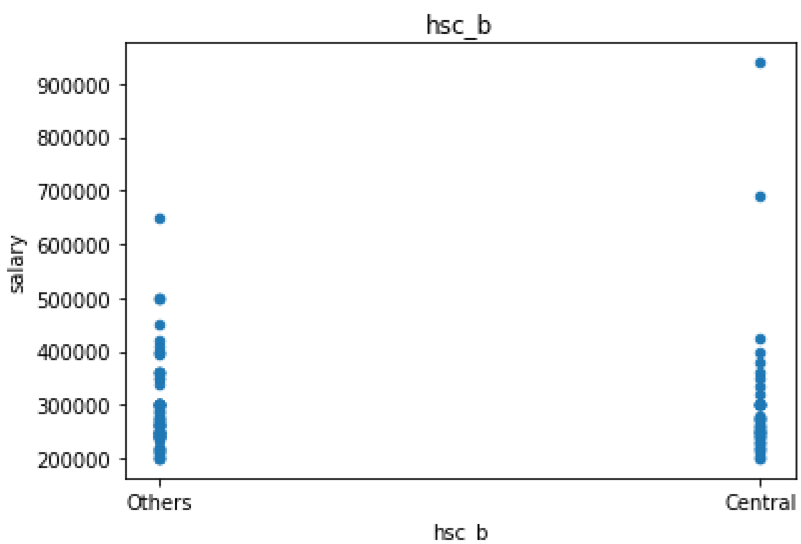

sl_no
900000
800000
700000
600000
500000
400000
300000
200000
salary
0
50
100
150
200
sl_no

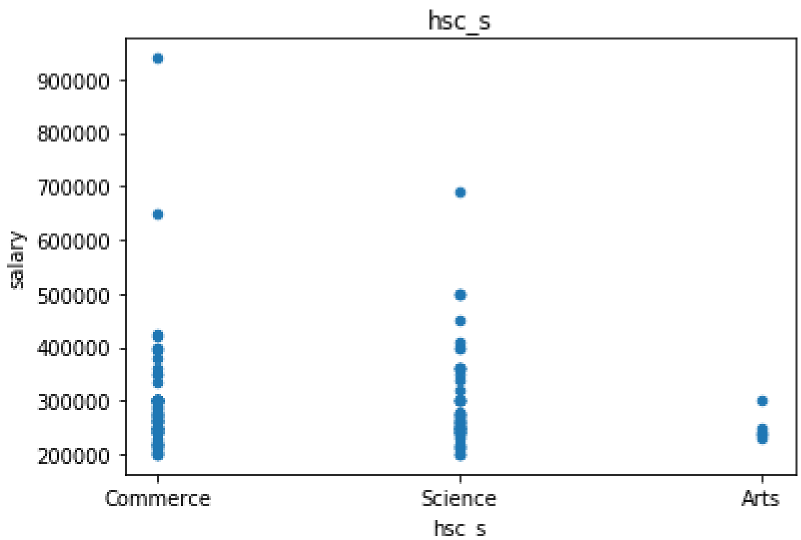

gender
900000
800000
700000
600000
500000
400000
300000
200000
salary
M
F
gender

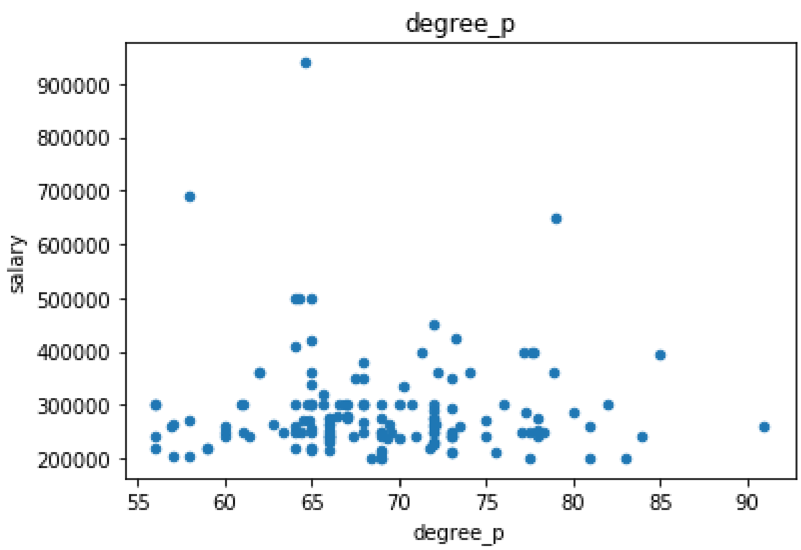

ssc_p
900000
800000
700000
600000
500000
400000
300000
200000
salary
50
55
60
65
70
75
80
85
90
ssc_p

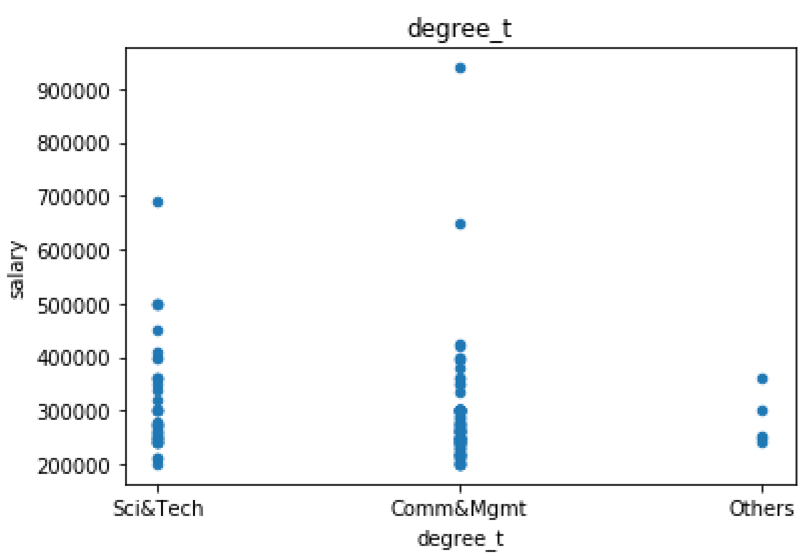

ssc_b
salary
900000
800000
700000
600000
500000
400000
300000
200000
Others
Central
ssc_b

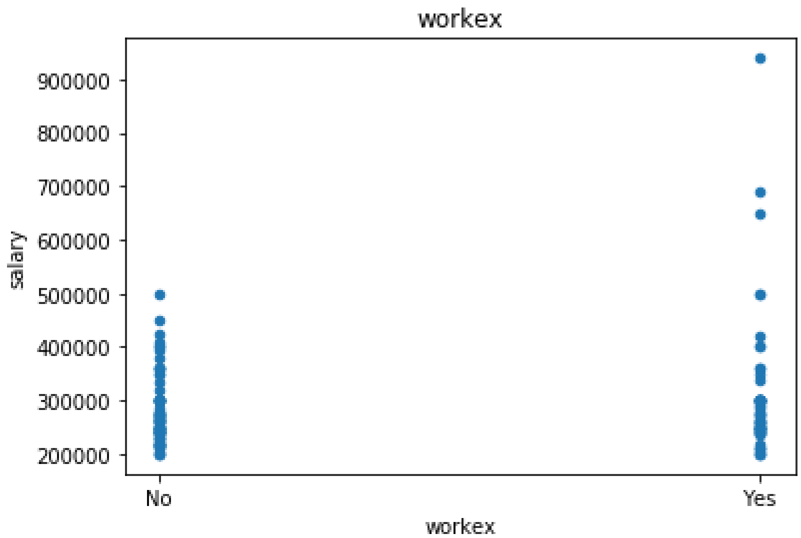

hsc_p
salary
900000
800000
700000
600000
500000
400000
300000
200000
50
60
70
80
90
100
hsc_p

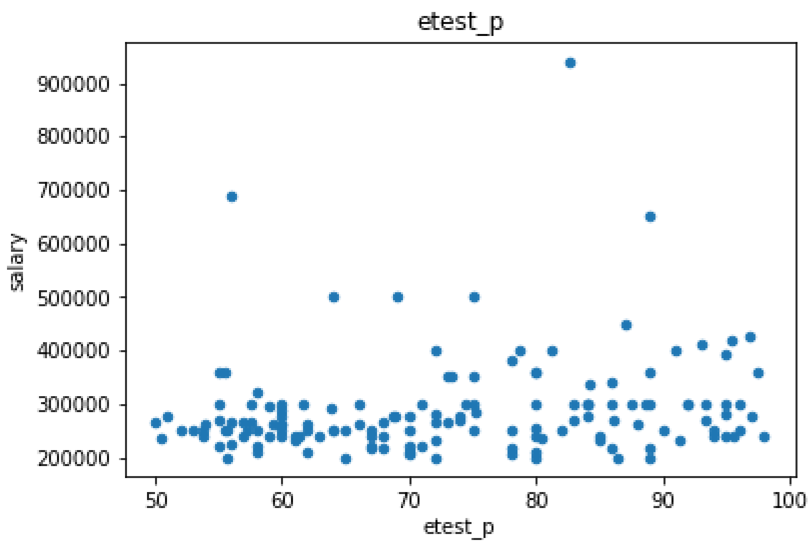

hsc_b
salary
900000
800000
700000
600000
500000
400000
300000
200000
Others
Central
hsc_b

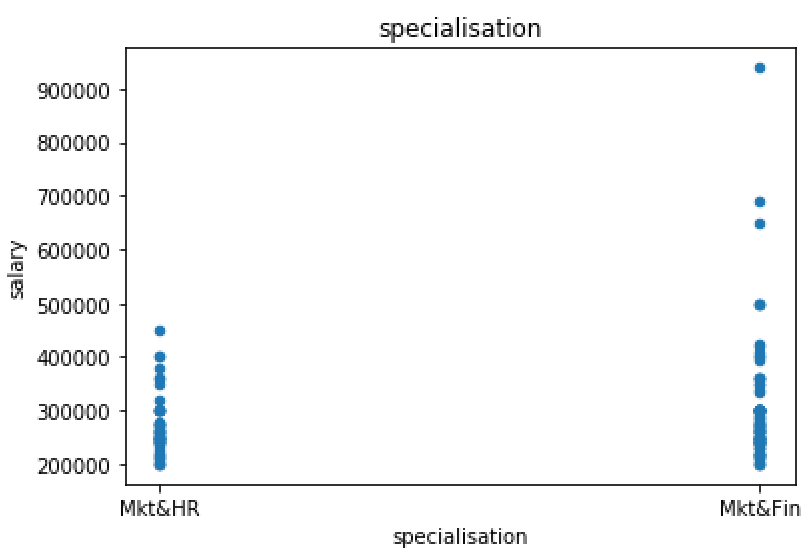

hsc_s
900000
800000
700000
600000
500000
400000
300000
200000
salary
Commerce
Science
Arts
hsc_s

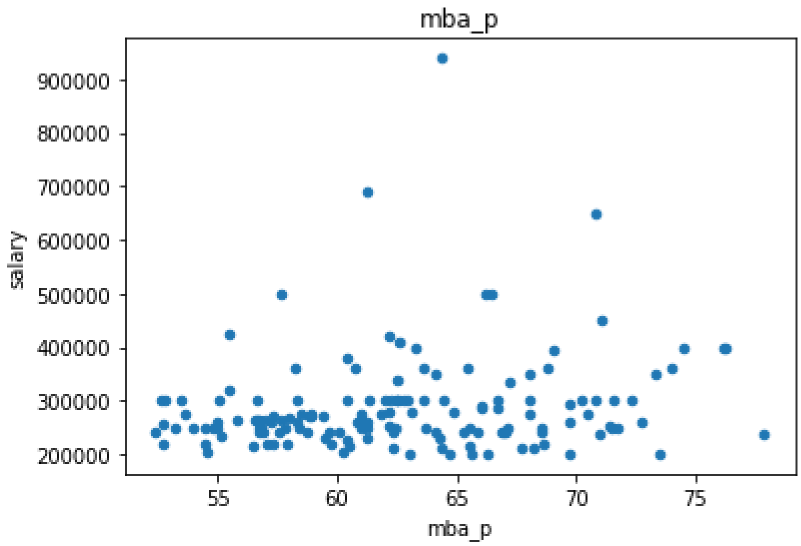

degree_p
900000
800000
700000
600000
500000
400000
300000
200000
salary
55
60
65
70
75
80
85
90
degree_p

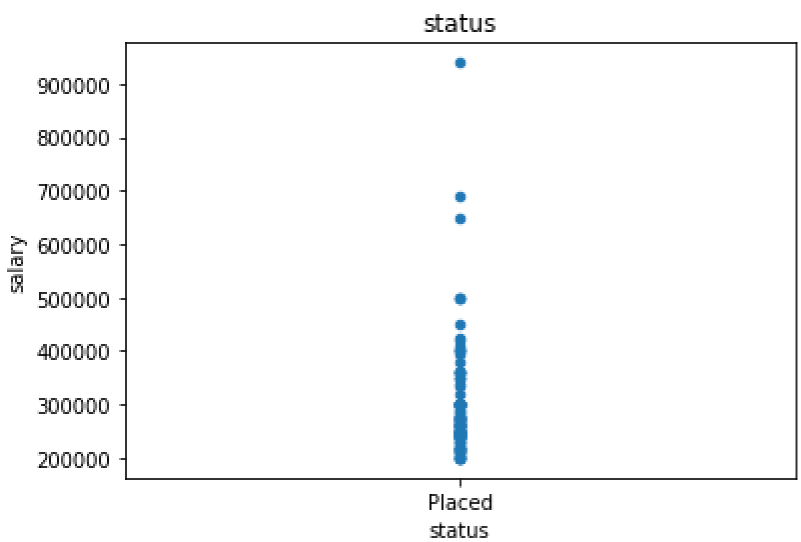

degree_t
900000
800000
700000
600000
500000
400000
300000
200000
salary
Sci&Tech
Comm&Mgmt
Others
degree_t

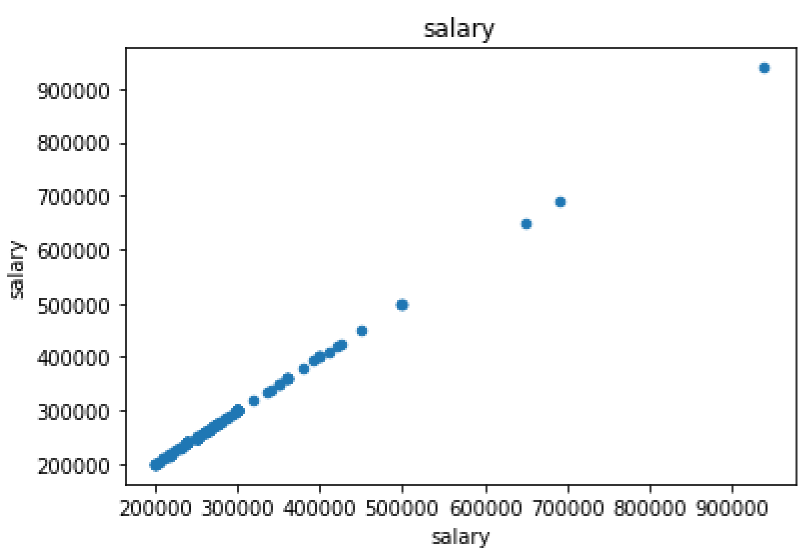

workex
900000
800000
700000
600000
500000
400000
300000
200000
salary
No
Yes
workex


etest_p
900000
800000
700000
600000
500000
400000
300000
200000
salary
50
60
70
80
90
100
etest_p


specialisation
900000
800000
700000
600000
500000
400000
300000
200000
salary
Mkt&HR
Mkt&Fin
specialisation







After this EDA we can determine many things, especially the distribution of the salary variable and that we have many Categorical Variables that we have to onehotencode afterwards in the preprocessing

Data Preprocessing

Counting the amount of NaN values for each feature

In [7]:
```
for c in df.columns:
    nans=df[c].isna().sum()
    print(c,': ',nans)
```

```
sl_no :  0
gender :  0
ssc_p :  0
ssc_b :  0
hsc_p :  0
hsc_b :  0
hsc_s :  0
degree_p :  0
degree_t :  0
workex :  0
etest_p :  0
specialisation :  0
mba_p :  0
status :  0
salary :  67
```

Here we can see that we have 67 NaN values in the salary feature out of the 215 values (a 31%)

In [8]:
```
print(len(df['status'].where(df['status']=='Not Placed') .dropna()))
```

```
67
```

But the previous 67 NaN salary values correspond to the Not Place so we are going to assume the have 0.0 salary

In [9]:
```
df['salary']=[df['salary'].iloc[i] if df['status'].iloc[i]!='Not Placed' else 0.0 for i in range(len(df))]
df
```

Out[9]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No
...	...	...	...	...	...	...	...	...	...	...
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	No
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	No
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes
213	214	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	No
214	215	M	62.00	Central	58.00	Others	Science	53.00	Comm&Mgmt	No

215 rows × 15 columns

Moving forward we will change the features in workex to binary (No,Yes)->(0,1)

In [10]:
```
df['workex']=[1 if w=='Yes' else 0 for w in df['workex']]
df.head()
```

Out[10]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	e
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	0	
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	1	
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	0	
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	0	
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	0	

we will change the features in status to binary (Not Placed,Placed)->(0,1)

In [11]:
```
df['status']=[1 if w=='Placed' else 0 for w in df['status']]
df.head()
```

Out[11]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	et
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	0	
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	1	
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	0	
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	0	
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	0	

Now we turn Gender into a binary variable (F,M)->(0,1)

In [12]:
```
df['gender']=[1 if w=='M' else 0 for w in df['gender']]
df.head()
```

Out[12]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	et
0	1	1	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	0	
1	2	1	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	1	
2	3	1	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	0	
3	4	1	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	0	
4	5	1	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	0	

We are going to change ssc_p to a binary (others,central)->(0,1)

In [13]:
```
df['ssc_b']=[1 if w=='Central' else 0 for w in df['ssc_b']]
df.head()
```

Out[13]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	et
0	1	1	67.00	0	91.00	Others	Commerce	58.00	Sci&Tech	0	
1	2	1	79.33	1	78.33	Others	Science	77.48	Sci&Tech	1	
2	3	1	65.00	1	68.00	Central	Arts	64.00	Comm&Mgmt	0	
3	4	1	56.00	1	52.00	Central	Science	52.00	Sci&Tech	0	
4	5	1	85.80	1	73.60	Central	Commerce	73.30	Comm&Mgmt	0	

We are going to change hsc_b to a binary (others,central)->(0,1)

In [14]:
```
df['hsc_b']=[1 if w=='Central' else 0 for w in df['hsc_b']]
df.head()
```

Out[14]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	ete
0	1	1	67.00	0	91.00	0	Commerce	58.00	Sci&Tech	0	
1	2	1	79.33	1	78.33	0	Science	77.48	Sci&Tech	1	
2	3	1	65.00	1	68.00	1	Arts	64.00	Comm&Mgmt	0	
3	4	1	56.00	1	52.00	1	Science	52.00	Sci&Tech	0	
4	5	1	85.80	1	73.60	1	Commerce	73.30	Comm&Mgmt	0	

Now onehot encode Hsc_s ('Commerce' 'Science' 'Arts')->(0,1,2)

In [15]:
```
hsc_s_dict=dict(zip(df['hsc_s'].unique(),range(3)))
df['hsc_s']=[hsc_s_dict[h] for h in df['hsc_s']]
df.head()
```

Out[15]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p
0	1	1	67.00	0	91.00	0	0	58.00	Sci&Tech	0	55.0
1	2	1	79.33	1	78.33	0	1	77.48	Sci&Tech	1	86.5
2	3	1	65.00	1	68.00	1	2	64.00	Comm&Mgmt	0	75.0
3	4	1	56.00	1	52.00	1	1	52.00	Sci&Tech	0	66.0
4	5	1	85.80	1	73.60	1	0	73.30	Comm&Mgmt	0	96.8

Now onehot encode degree_t ('Sci&Tech' 'Comm&Mgmt' 'Others')->(0,1,2)

In [16]:
```
degree_t_dict=dict(zip(df['degree_t'].unique(),range(3)))
df['degree_t']=[degree_t_dict[h] for h in df['degree_t']]
df.head()
```

Out[16]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	s
0	1	1	67.00	0	91.00	0	0	58.00	0	0	55.0	
1	2	1	79.33	1	78.33	0	1	77.48	0	1	86.5	
2	3	1	65.00	1	68.00	1	2	64.00	1	0	75.0	
3	4	1	56.00	1	52.00	1	1	52.00	0	0	66.0	
4	5	1	85.80	1	73.60	1	0	73.30	1	0	96.8	

In [17]:
```
specialisation_dict=dict(zip(df['specialisation'].unique(),range(2)))
df['specialisation']=[specialisation_dict[h] for h in df['specialisation']]
df.head()
```

Out[17]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	s
0	1	1	67.00	0	91.00	0	0	58.00	0	0	55.0	
1	2	1	79.33	1	78.33	0	1	77.48	0	1	86.5	
2	3	1	65.00	1	68.00	1	2	64.00	1	0	75.0	
3	4	1	56.00	1	52.00	1	1	52.00	0	0	66.0	
4	5	1	85.80	1	73.60	1	0	73.30	1	0	96.8	

We just ended the preprocessing part by changing the Nan in salary to 0.0 and onehot encoding the categorical variables But for obvious purposes we will remove the serial number feature

In [18]:
```
df=df[df.columns[1:]]
df.head()
```

Out[18]:

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisa
0	1	67.00	0	91.00	0	0	58.00	0	0	55.0	
1	1	79.33	1	78.33	0	1	77.48	0	1	86.5	
2	1	65.00	1	68.00	1	2	64.00	1	0	75.0	
3	1	56.00	1	52.00	1	1	52.00	0	0	66.0	
4	1	85.80	1	73.60	1	0	73.30	1	0	96.8	

Second EDA (After Preprocessing)

In [19]:
```
df.describe()
```

Out[19]:

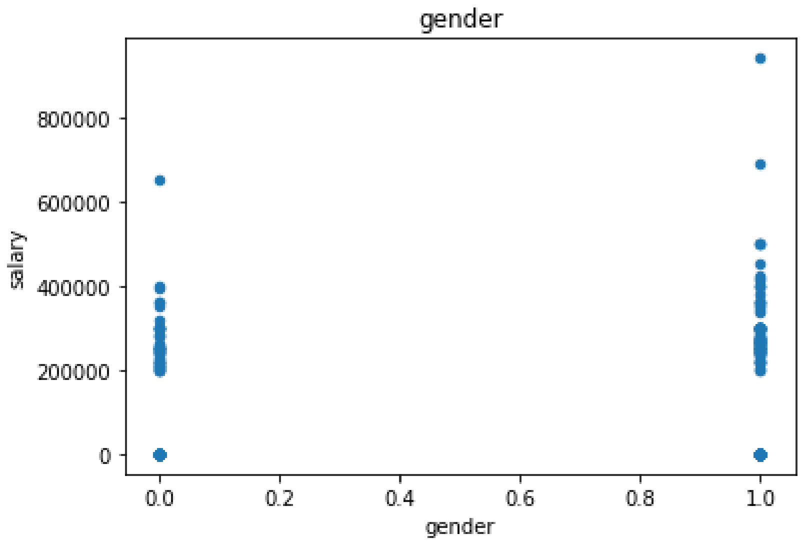
	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	d
count	215.000000	215.000000	215.000000	215.000000	215.000000	215.000000	215.000000	215
mean	0.646512	67.303395	0.539535	66.333163	0.390698	0.525581	66.370186	0
std	0.479168	10.827205	0.499598	10.897509	0.489045	0.594403	7.358743	0
min	0.000000	40.890000	0.000000	37.000000	0.000000	0.000000	50.000000	0
25%	0.000000	60.600000	0.000000	60.900000	0.000000	0.000000	61.000000	0
50%	1.000000	67.000000	1.000000	65.000000	0.000000	0.000000	66.000000	1
75%	1.000000	75.700000	1.000000	73.000000	1.000000	1.000000	72.000000	1
max	1.000000	89.400000	1.000000	97.700000	1.000000	2.000000	91.000000	2

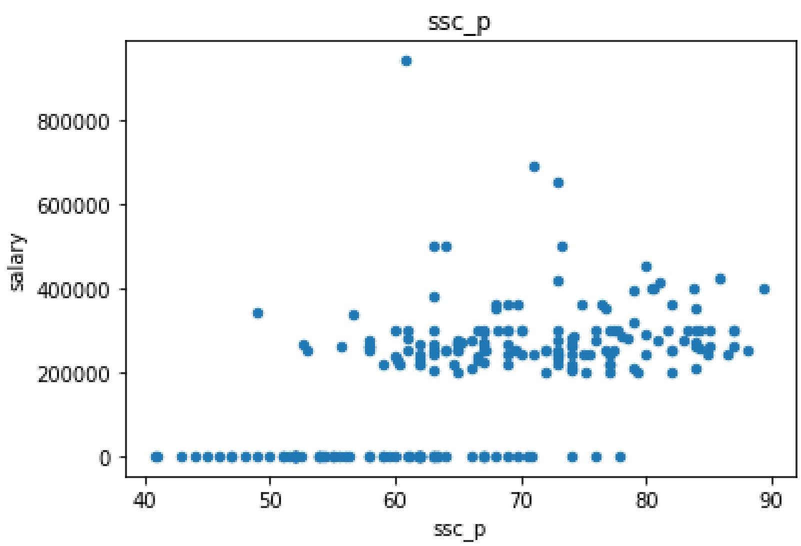
In [20]:
```
df.corr()['salary']
```

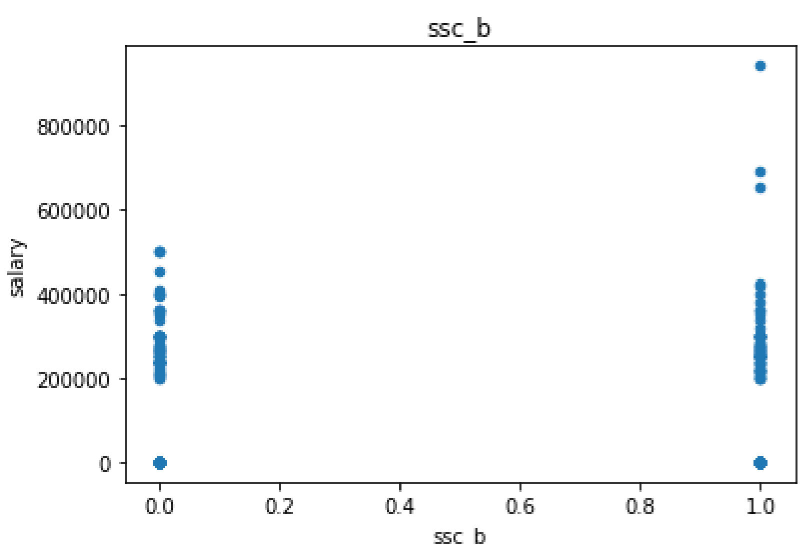
Out[20]:
```
gender            0.143110
ssc_p             0.538090
ssc_b            -0.034594
hsc_p             0.452569
hsc_b            -0.011544
hsc_s            -0.048067
degree_p          0.408371
degree_t         -0.112384
workex            0.298285
etest_p           0.186988
specialisation    0.275766
mba_p             0.139823
status            0.865774
salary            1.000000
Name: salary, dtype: float64
```

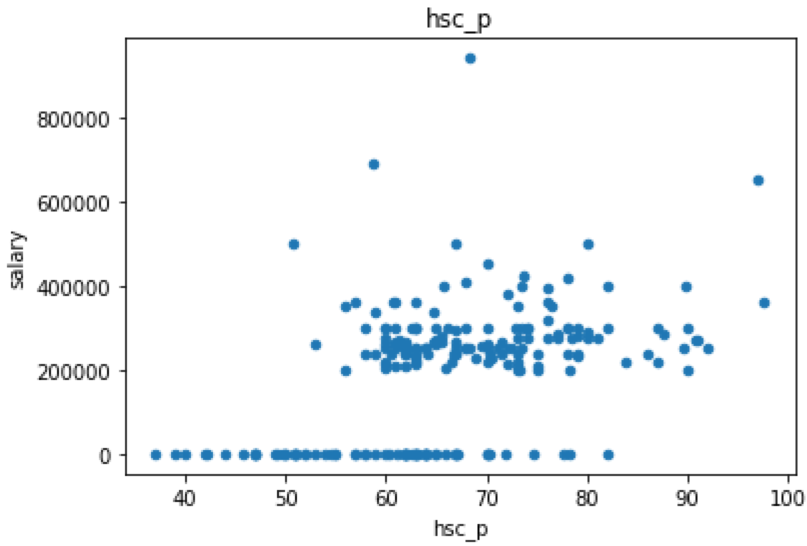
After the first part of preprocessing we can now determine that there are many variables that correleate with our varibale of interest (salary)

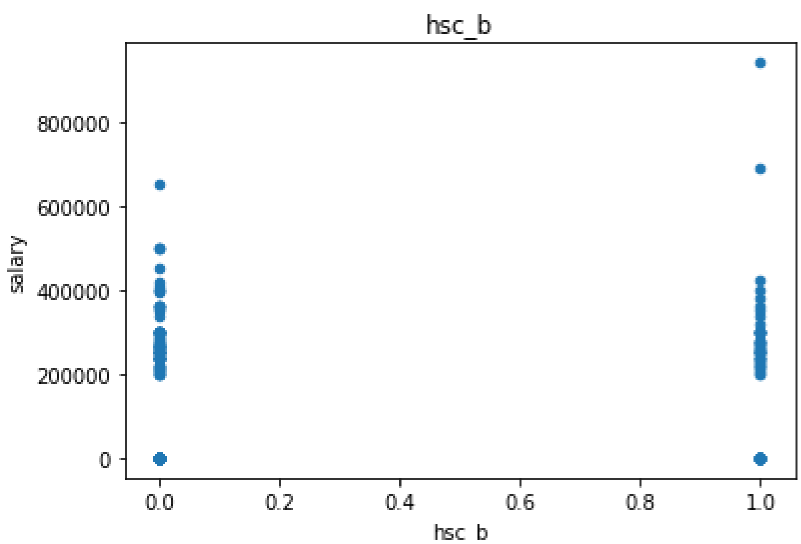
In [21]:
```
for c in df.columns:
    try:
        df.plot(c,'salary',kind='scatter',title=c)
    except:
        pass
```

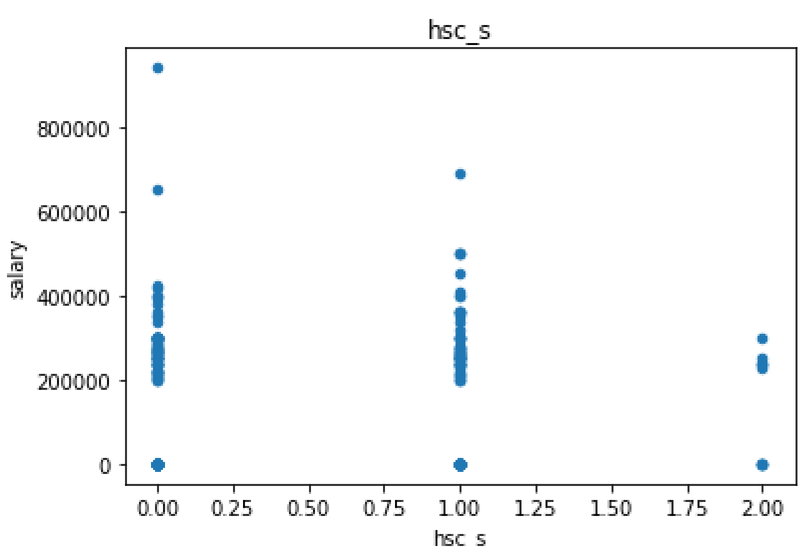

gender
salary
800000
600000
400000
200000
0
0.0
0.2
0.4
0.6
0.8
1.0
gender

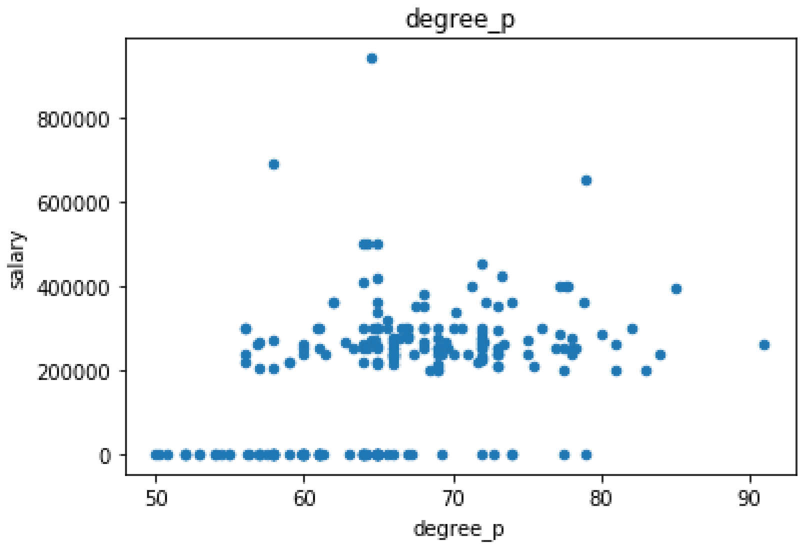

ssc_p
salary
800000
600000
400000
200000
0
40
50
60
70
80
90
ssc_p

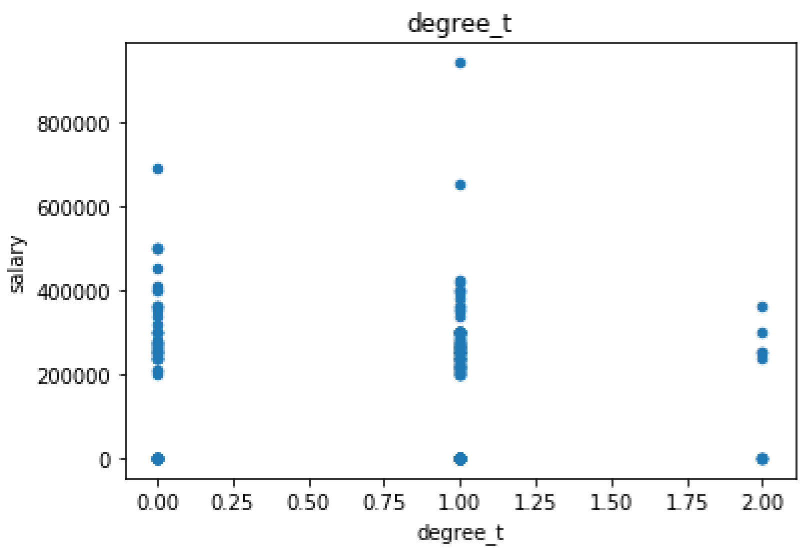

ssc_b
salary
800000
600000
400000
200000
0
0.0
0.2
0.4
0.6
0.8
1.0
ssc_b

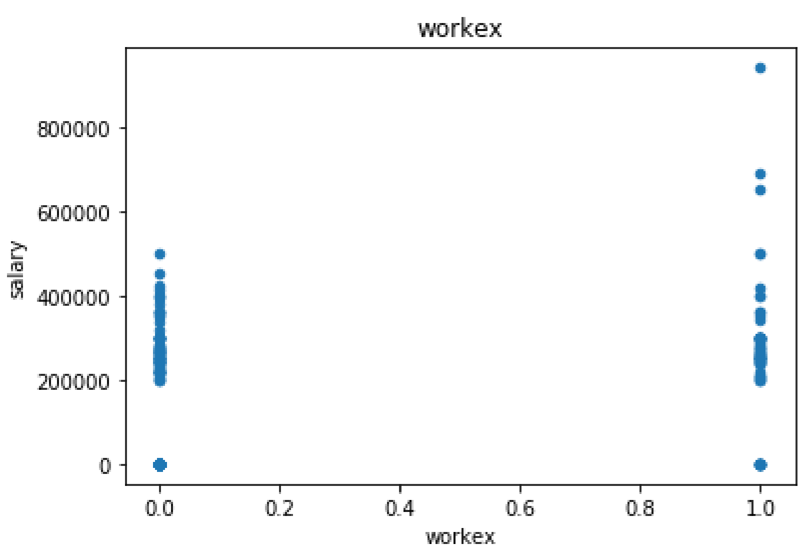

hsc_p
salary
hsc_p
0
200000
400000
600000
800000
40
50
60
70
80
90
100

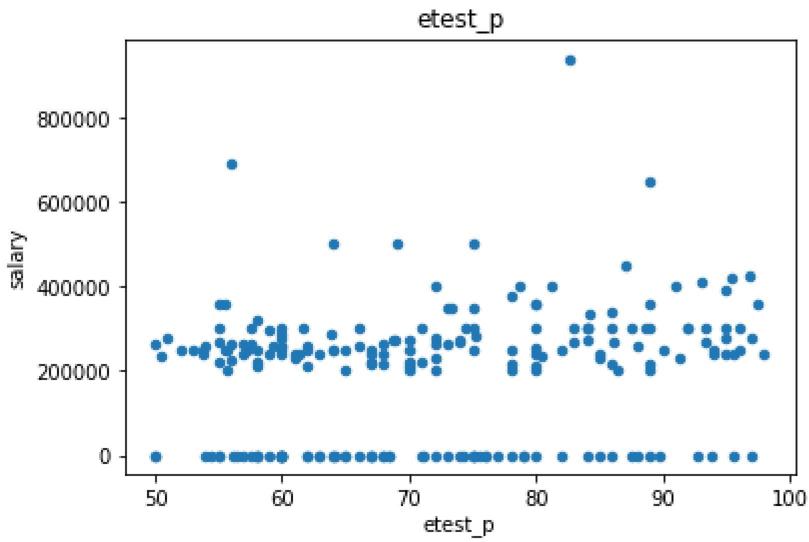

hsc_b
salary
hsc_b
0
200000
400000
600000
800000
0.0
0.2
0.4
0.6
0.8
1.0

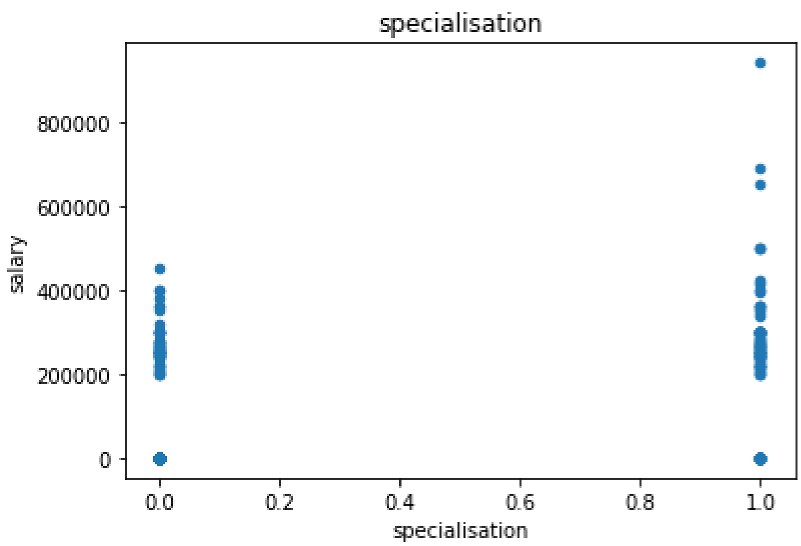

hsc_s
salary
hsc_s
0
200000
400000
600000
800000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

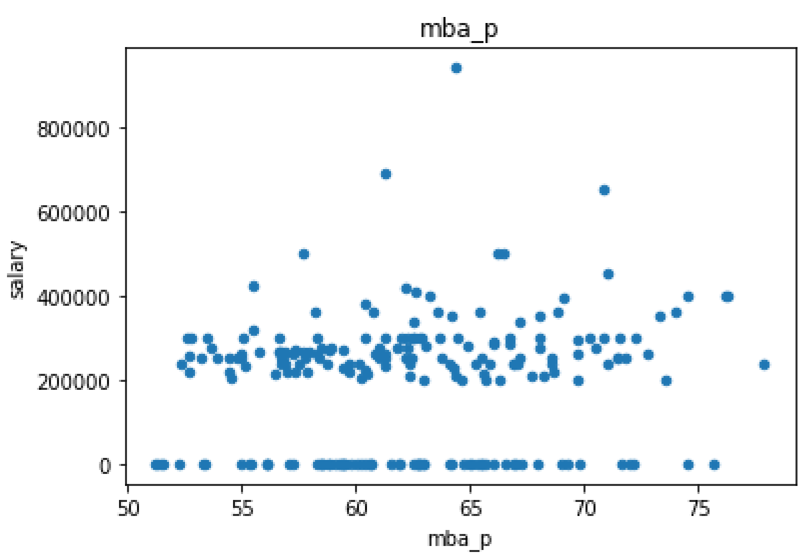

degree_p
800000
600000
400000
200000
0
salary
50
60
70
80
90
degree_p

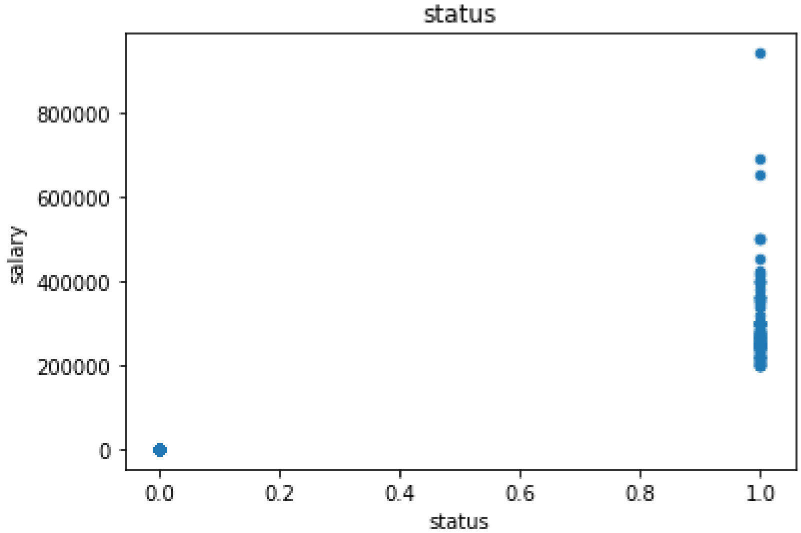

degree_t
800000
600000
400000
200000
0
salary
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
degree_t

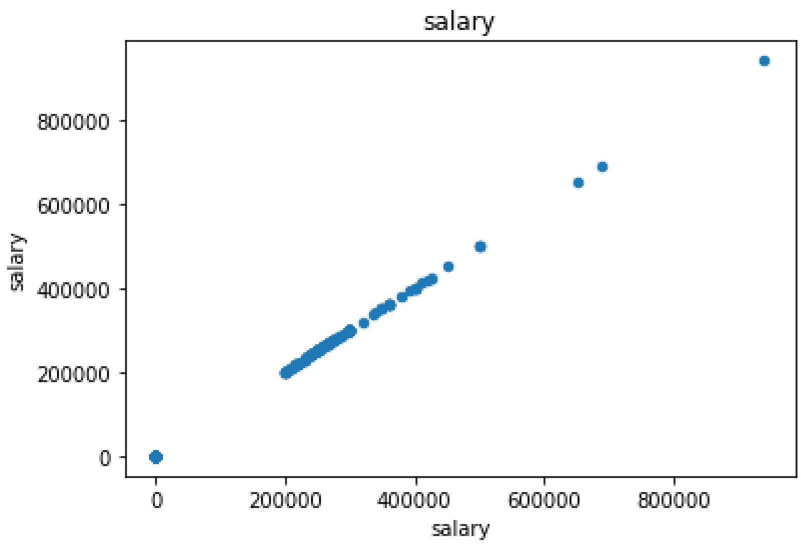

workex
800000
600000
400000
200000
0
salary
0.0
0.2
0.4
0.6
0.8
1.0
workex


etest_p
salary
800000
600000
400000
200000
0
50
60
70
80
90
100
etest_p


specialisation
salary
800000
600000
400000
200000
0
0.0
0.2
0.4
0.6
0.8
1.0
specialisation


mba_p
salary
800000
600000
400000
200000
0
50
55
60
65
70
75
mba_p



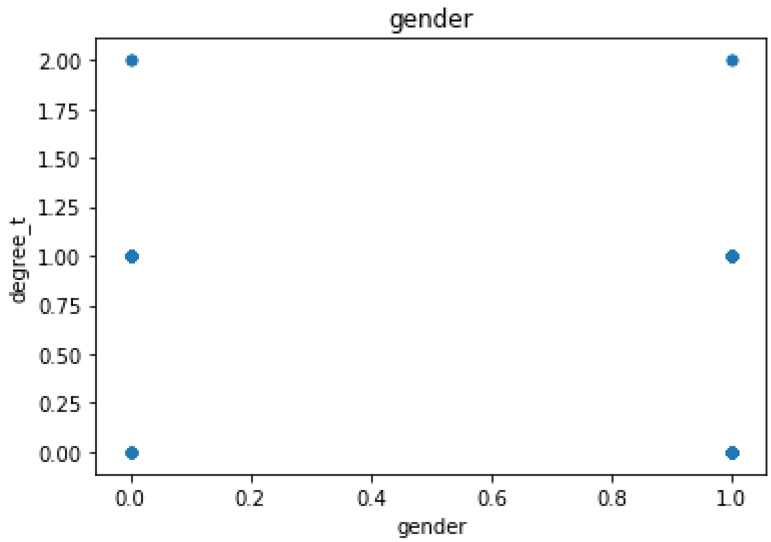


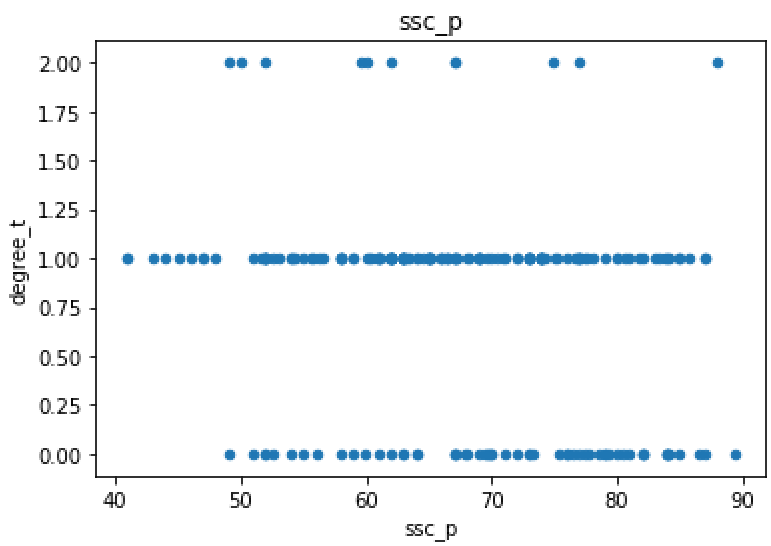
In [22]:
```
df.corr()['degree_t']
```

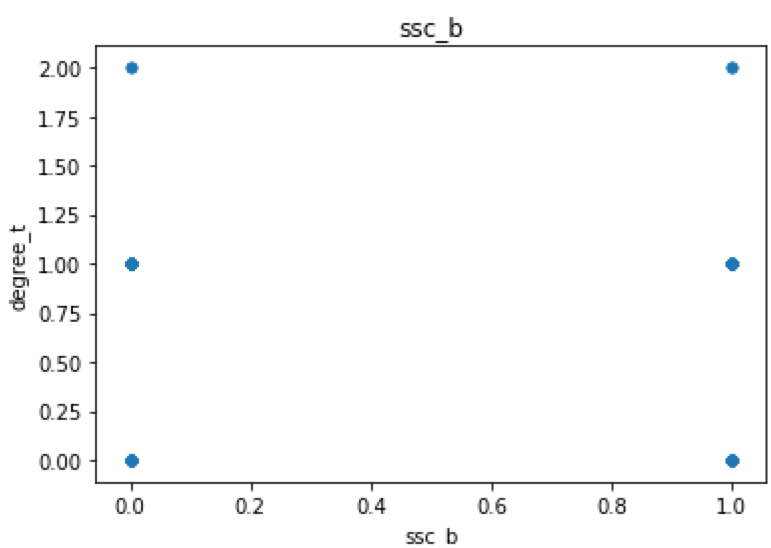
Out[22]:
```
gender            -0.110567
ssc_p             -0.215745
ssc_b              0.087035
hsc_p             -0.009579
hsc_b              0.122604
hsc_s             -0.250514
degree_p          -0.180624
degree_t           1.000000
workex            -0.083505
etest_p           -0.005386
specialisation     0.014103
mba_p             -0.121357
status            -0.056572
salary            -0.112384
Name: degree_t, dtype: float64
```

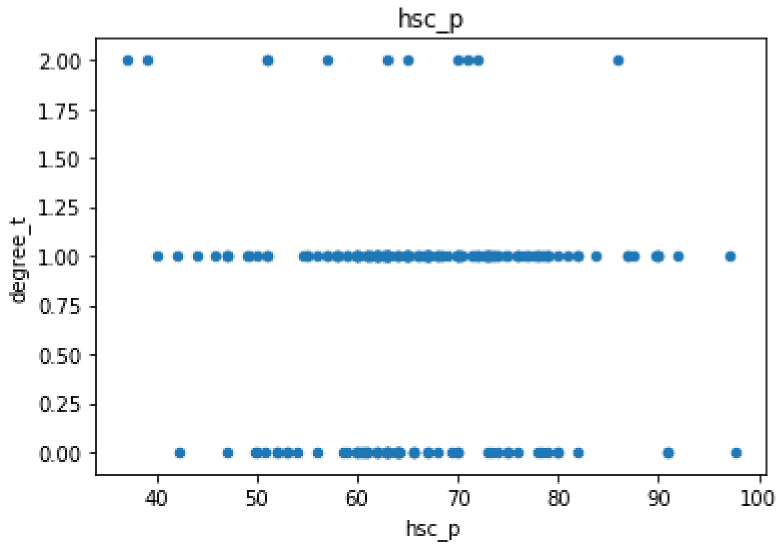
After the first part of preprocessing we can now determine that there are many variables that correleate with our varibale of interest (degree)

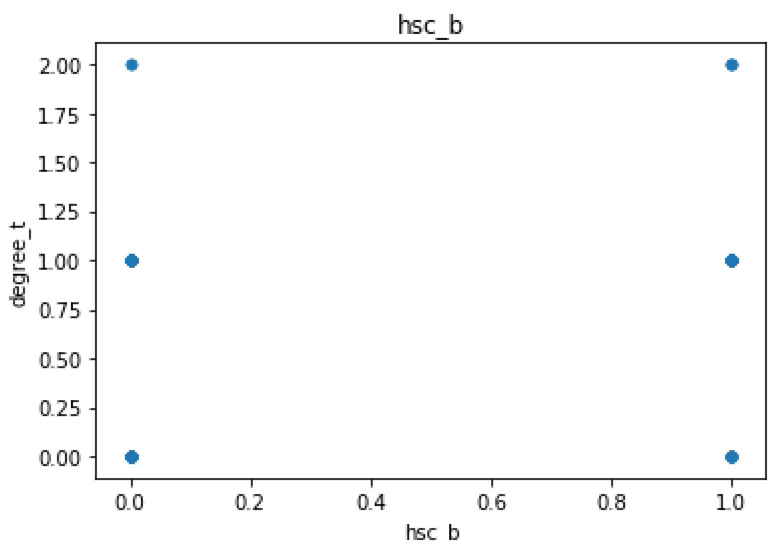
In [23]:
```
for c in df.columns:
    try:
        df.plot(c,'degree_t',kind='scatter',title=c)
    except:
        pass
```

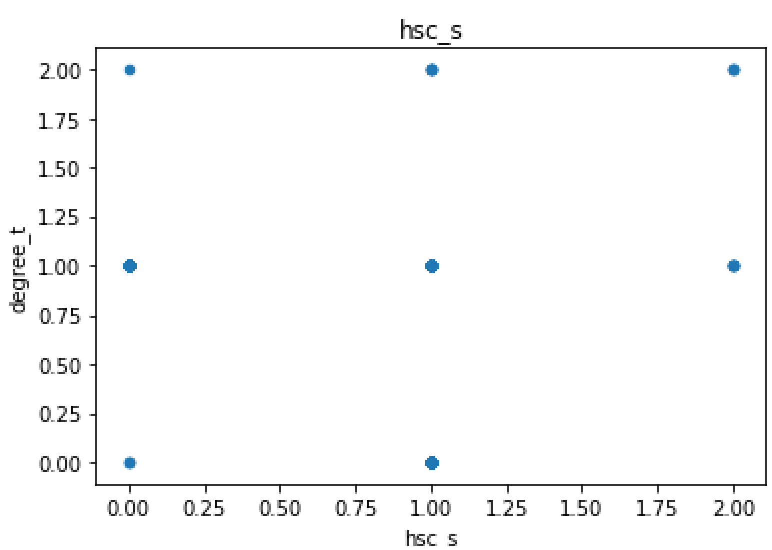

gender
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
degree_t
0.0
0.2
0.4
0.6
0.8
1.0
gender

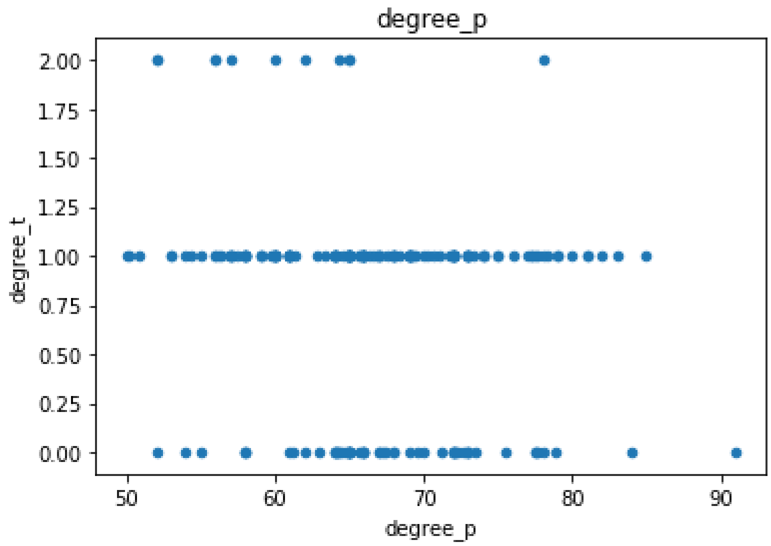

ssc_p
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
degree_t
40
50
60
70
80
90
ssc_p

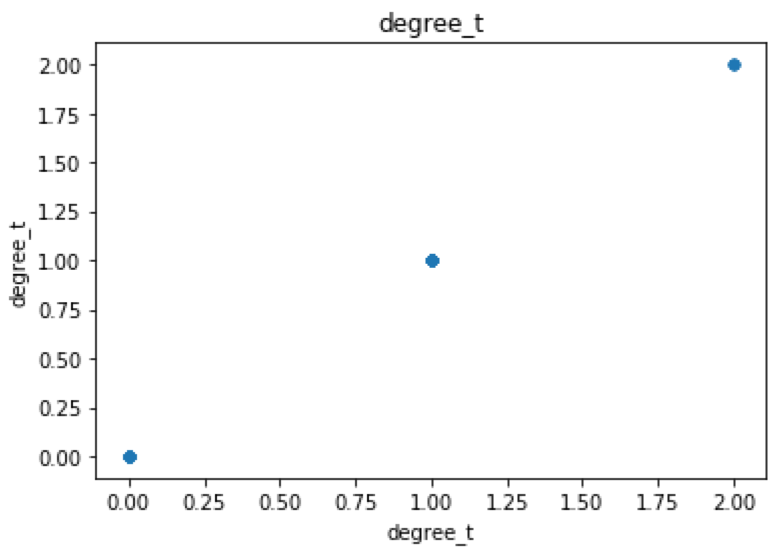

ssc_b
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
degree_t
0.0
0.2
0.4
0.6
0.8
1.0
ssc_b

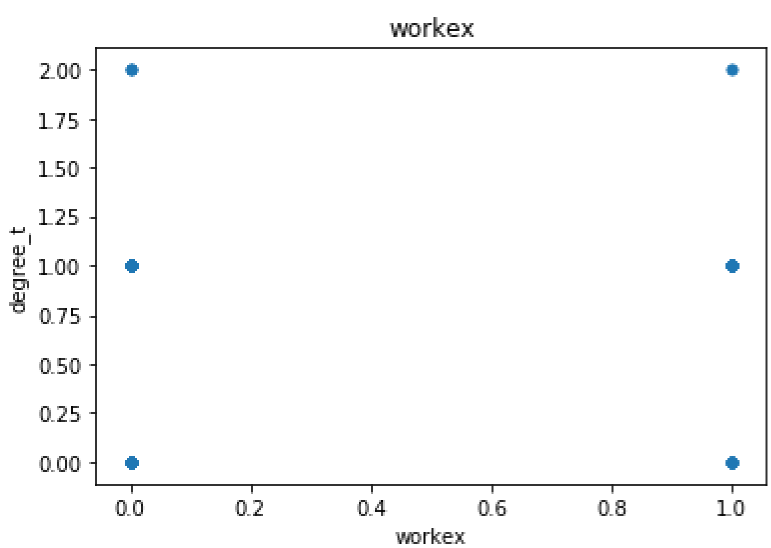

hsc_p
degree_t
hsc_p
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
40
50
60
70
80
90
100

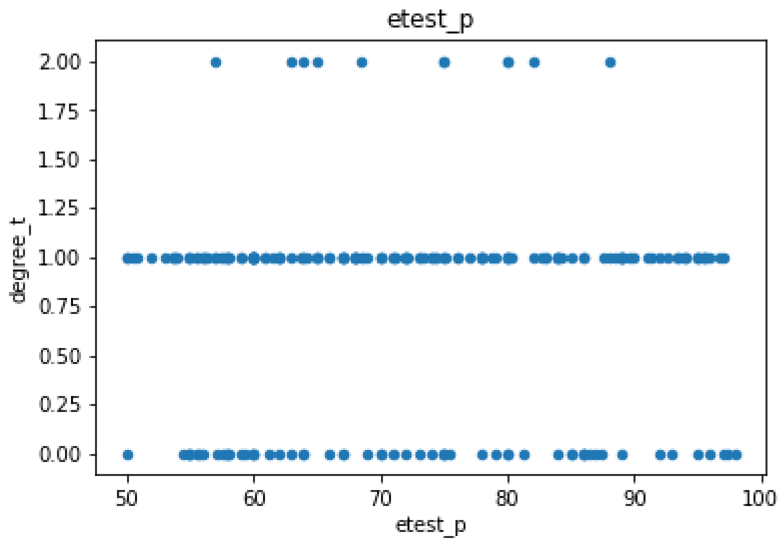

hsc_b
degree_t
hsc_b
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8
1.0

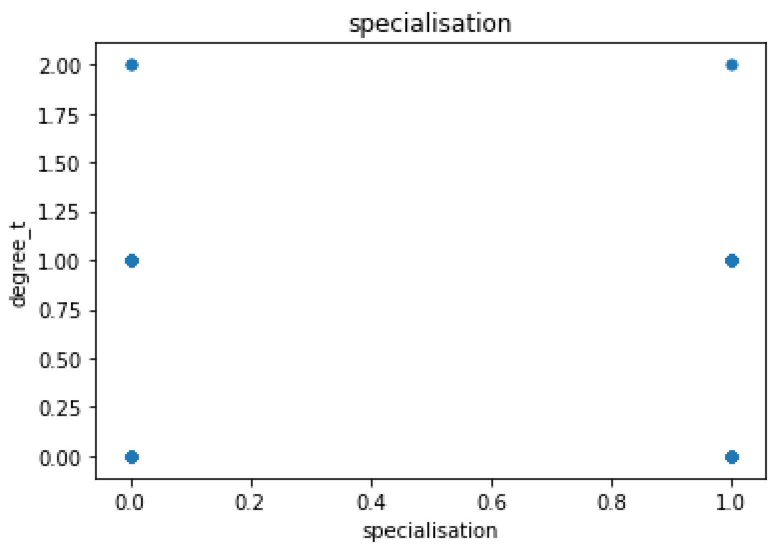

hsc_s
degree_t
hsc_s
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

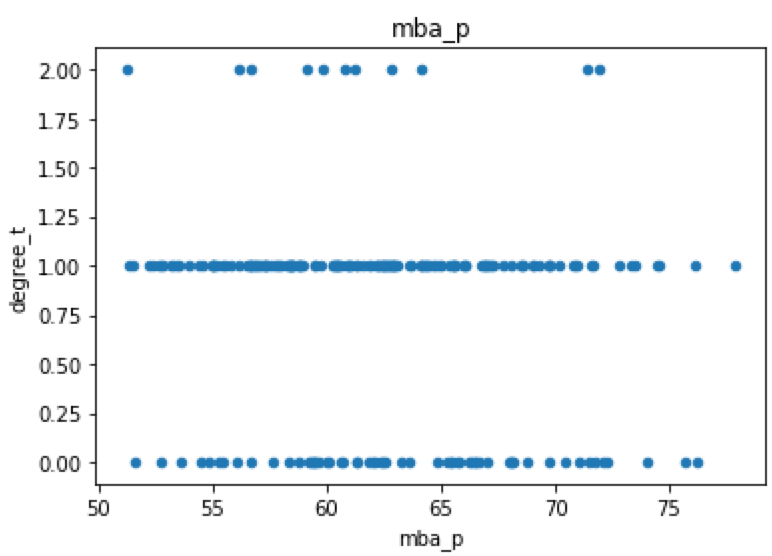

degree_p
degree_t
degree_p

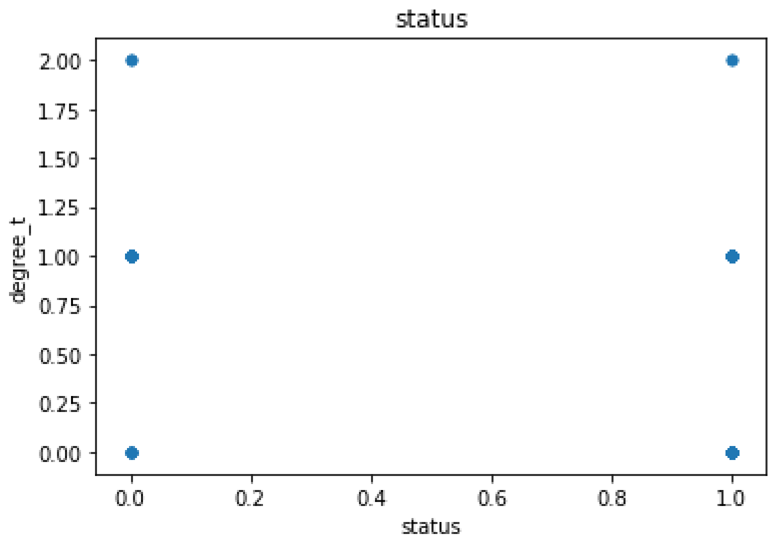

degree_t
degree_t
degree_t

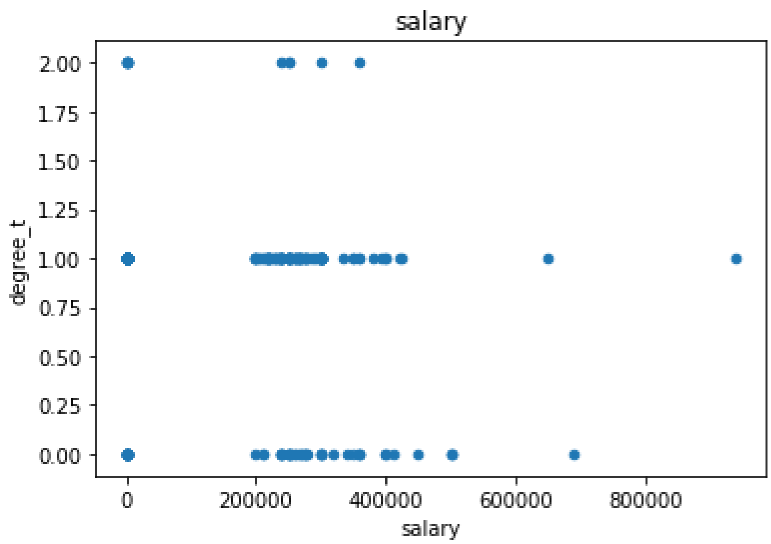

workex
degree_t
workex


etest_p
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
degree_t
50
60
70
80
90
100
etest_p


specialisation
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
degree_t
0.0
0.2
0.4
0.6
0.8
1.0
specialisation


mba_p
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
degree_t
50
55
60
65
70
75
mba_p





Models

we are going to do models for two features: salary (predctive), degree(classification).

For salary we are going to do linear models such that

$$\vec{\theta} \cdot \vec{\mathbf{X}} = Salary$$

And for the degree

$$\vec{\theta} \cdot \vec{\mathbf{X}} = Degree$$

We have to create our test and training sets

In [24]:
```
train,test=train_test_split(df,test_size=0.2,random_state=42)
```

Prediction model for Salary

We are going to start with the models for salary Linear models from 1-N dimensions based on the 4 features with the highest correlations

$$\vec{\theta} \cdot \overrightarrow{\mathbf{status}} = Salary$$

In [25]:
```
univarSalaryStatus=LinearRegression().fit(train['status'].values.reshape((-1,1
)),train['salary'])
univarSalaryStatus.coef_
```

Out[25]:
```
array([289914.52991453])
```

In [26]:
```
print('the Score of the model is: ', univarSalaryStatus.score(test['status'].v
alues.reshape((-1,1)),test['salary']))
```

```
the Score of the model is:  0.7341180253269654
```

In this first model we can see that accuracy is 'high' mostly due to the fact that having a job implies that you have a salary

the second model based on the ssc_p

$$\vec{\theta} \cdot \overrightarrow{\mathbf{sscp}} = Salary$$

In [27]:
```
univarSalarySsc_p=LinearRegression().fit(train['ssc_p'].values.reshape((-1,1
)),train['salary'])
univarSalarySsc_p.coef_
```

Out[27]:
```
array([7939.25344376])
```

In [28]:
```
print('the Score of the model is: ', univarSalarySsc_p.score(test['ssc_p'].val
ues.reshape((-1,1)),test['salary']))
```

```
the Score of the model is:  0.2136818306435675
```

In this model the accuracy dwindle to ~1/4 of the accuracy of the previous model

the second model based on the hsc_p

$$\vec{\theta} \cdot \overrightarrow{\mathbf{hscp}} = Salary$$

In [29]:
```
univarSalaryHsc_p=LinearRegression().fit(train['hsc_p'].values.reshape((-1,1
)),train['salary'])
univarSalaryHsc_p.coef_
```

Out[29]:
```
array([6714.95850694])
```

In [30]:
```
print('the Score of the model is: ', univarSalaryHsc_p.score(test['hsc_p'].val
ues.reshape((-1,1)),test['salary']))
```

```
the Score of the model is:  0.029599473052214442
```

Now in this last model the accuracy is even less than 10%

Now we build multivariate models

we are going

$$\vec{\theta} \cdot \overrightarrow{\mathbf{features}} = Salary$$

In [31]:
```
orderedCorrelations=abs(df.corr()['salary']).sort_values(ascending=False).keys
()
maxScore=0
bestCombination=[]
for sizeComb in range(2,len(orderedCorrelations)-1):
    for comb in combinations(orderedCorrelations[1:], sizeComb):
        model=LinearRegression().fit(train[list(comb)],train['salary'])
        score=model.score(test[list(comb)],test['salary'])
        print(comb,' : ',score)
        if(score>maxScore):
            maxScore=score
            bestCombination=comb
```

```
p', 'gender', 'mba_p', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7185455102478817
('status', 'ssc_p', 'hsc_p', 'degree_p', 'workex', 'specialisation', 'etest_
p', 'gender', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7155603589862517
('status', 'ssc_p', 'hsc_p', 'degree_p', 'workex', 'specialisation', 'etest_
p', 'mba_p', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7142813001061445
('status', 'ssc_p', 'hsc_p', 'degree_p', 'workex', 'specialisation', 'gende
r', 'mba_p', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7535291479010588
('status', 'ssc_p', 'hsc_p', 'degree_p', 'workex', 'etest_p', 'gender', 'mba_
p', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7263606631324367
('status', 'ssc_p', 'hsc_p', 'degree_p', 'specialisation', 'etest_p', 'gende
r', 'mba_p', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7219401345802416
('status', 'ssc_p', 'hsc_p', 'workex', 'specialisation', 'etest_p', 'gender',
'mba_p', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7211475300614182
('status', 'ssc_p', 'degree_p', 'workex', 'specialisation', 'etest_p', 'gende
r', 'mba_p', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7287410170757171
('status', 'hsc_p', 'degree_p', 'workex', 'specialisation', 'etest_p', 'gende
r', 'mba_p', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.7435074120417804
('ssc_p', 'hsc_p', 'degree_p', 'workex', 'specialisation', 'etest_p', 'gende
r', 'mba_p', 'degree_t', 'hsc_s', 'ssc_b', 'hsc_b')  :  0.34236523836886784
```

In [32]:
```
print('Best Combination ',bestCombination,' : ',maxScore)
```

```
Best Combination  ('status', 'degree_p', 'workex', 'gender', 'mba_p', 'degree
_t', 'ssc_b', 'hsc_b')  :  0.7703994663290581
```

The combination see in the cell above is the best feature combination for the linear regression in order to predict the salary

Classification model for Degree with tensorflow

In [33]:
```
model = tf.keras.models.Sequential([tf.keras.layers.Dense(1),
                                    tf.keras.layers.Dense(3)])
model.compile(optimizer = tf.keras.optimizers.Adam(),
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])
xs,ys=train['hsc_s'].copy(),train['degree_t']
xs=xs/100
model.fit(xs.values.reshape((-1,1)), ys.values.reshape((-1,1)), epochs=500)
```

```
Epoch 495/500
172/172 [==============================] - 0s 41us/sample - loss: 0.3612 - ac
c: 0.4884
Epoch 496/500
172/172 [==============================] - 0s 46us/sample - loss: 0.3610 - ac
c: 0.4884
Epoch 497/500
172/172 [==============================] - 0s 47us/sample - loss: 0.3609 - ac
c: 0.4884
Epoch 498/500
172/172 [==============================] - 0s 41us/sample - loss: 0.3610 - ac
c: 0.4884
Epoch 499/500
172/172 [==============================] - 0s 41us/sample - loss: 0.3609 - ac
c: 0.4884
Epoch 500/500
172/172 [==============================] - 0s 47us/sample - loss: 0.3609 - ac
c: 0.4884
43/43 [==============================] - 0s 535us/sample - loss: 0.3718 - ac
c: 0.4884
0.4883721
```

The model displayed did a ~19% better than the one before, but lets see if we can generate an N-dimensional model with a higher test accuracy

Now we will test the for all possible models of features in tf, by using combinatorics, for 500 epochs to find the one with the best accuracy over test in classification

In [37]:
```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('acc')>0.999 and logs.get('loss')<0.5):
            print("\nReached 99.9% accuracy so cancelling training!")
            self.model.stop_training = True

callbacks = myCallback()


orderedCorrelations=abs(df.corr()['degree_t']).sort_values(ascending=False).ke
ys()
maxScore=0
bestloss=float('inf')
bestCombination=[]
checked=[]
goodEnough=False
for sizeComb in range(2,len(orderedCorrelations)-1):
    for comb in combinations(orderedCorrelations[1:], sizeComb):
        print(comb)
        if(set(comb) not in checked):
            checked.append(set(comb))
            print('Case: ',comb,' #',len(checked))
            model = tf.keras.models.Sequential([tf.keras.layers.Dense(sizeComb
),tf.keras.layers.Dense(3)])
            model.compile(optimizer = tf.keras.optimizers.Adam(),loss = 'mse',
metrics=['accuracy'])
            xs,ys=train[list(comb)].copy()/1.0,train['degree_t']
            model.fit(xs.values.reshape((-1,sizeComb)), ys.values.reshape((-1,
1)), epochs=100,callbacks=[callbacks])
            loss,score=model.evaluate(test[list(comb)],test['degree_t'])
            if(score>maxScore and bestloss>loss):
                maxScore=score
                bestloss=loss
                bestCombination=comb
            if(score>0.999 and 0.5>loss):
                goodEnough=True
                print('Found a model good enough!!')
                break
    if(goodEnough):
        break
```

```
Epoch 32/100
172/172 [==============================] - 0s 198us/sample - loss: 0.9764 - a
cc: 1.0000
Epoch 33/100
172/172 [==============================] - 0s 186us/sample - loss: 0.8507 - a
cc: 1.0000
Epoch 34/100
172/172 [==============================] - 0s 186us/sample - loss: 0.7526 - a
cc: 1.0000
Epoch 35/100
172/172 [==============================] - 0s 203us/sample - loss: 0.6735 - a
cc: 1.0000
Epoch 36/100
172/172 [==============================] - 0s 186us/sample - loss: 0.6082 - a
cc: 1.0000
Epoch 37/100
172/172 [==============================] - 0s 174us/sample - loss: 0.5546 - a
cc: 1.0000
Epoch 38/100
172/172 [==============================] - 0s 192us/sample - loss: 0.5118 - a
cc: 1.0000
Epoch 39/100
 32/172 [====>.........................] - ETA: 0s - loss: 0.5070 - acc: 1.00
00
Reached 99.9% accuracy so cancelling training!
172/172 [==============================] - 0s 192us/sample - loss: 0.4797 - a
cc: 1.0000
43/43 [==============================] - 1s 20ms/sample - loss: 0.4758 - acc:
1.0000
Found a model good enough!!
```

In [38]:
```
print('The best tf model have the features', bestCombination,' and had an test
accuracy ', maxScore, ' and a MSE on the test is of  ',bestloss)
```

```
The best tf model have the features ('hsc_s', 'degree_p')  and had an test ac
curacy  1.0  and a MSE on the test is of   14.631838310596555
```

Conclusions

In the end of this notebook we learned that salary is indeed define by a great combination of factors, as seen before. This is coherent when we assume the salary in real life, which means that the salary of a person is not only define by academic features but other thing (that´s why we couldn´t reach an accuracy of 77%). Whereas for determining the type of degree ('Sci&Tech' 'Comm&Mgmt' 'Others') we found out a tensorflow model that actually did outstanding well with an accuracy of 1 over the test data and only a MSE of 0.92. In this case, this means that we can assume that the degree type can be determine by both the Specialization in Higher Secondary Education (hsc_s) and degree percentage (degree_p).