Математическая статистика

Практическое задание 5

В данном задании предлагается провести некоторое исследование модели линейной регрессии и критериев для проверки статистических гипотез, в частности применить этим модели к реальным данным.

Правила:

- Выполненную работу нужно отправить на почту `probability.diht@yandex.ru`, указав тему письма "`[номер группы] Фамилия Имя - Задание 5`". Квадратные скобки обязательны. Вместо `Фамилия Имя` нужно подставить свои фамилию и имя.
- Прислать нужно ноутбук и его pdf-версию. Названия файлов должны быть такими: `5.N.ipynb` и `5.N.pdf`, где `N` - ваш номер из таблицы с оценками.
- Никакой код из данного задания при проверке запускаться не будет.
- Некоторые задачи отмечены символом *. Эти задачи являются дополнительными. Успешное выполнение большей части таких задач (за все задания) является необходимым условием получения бонусного балла за практическую часть курса.
- Баллы за каждую задачу указаны далее. Если сумма баллов за задание меньше 25% (без учета доп. задач), то все задание оценивается в 0 баллов.

Баллы за задание:

- Задача 1 - 7 баллов
- Задача 2 - 2 балла
- Задача 3* - 3 балла
- Задача 4 - 2 балла
- Задача 5* - 10 баллов
- Задача 6 - 5 баллов
- Задача 7 - 4 балла
- Задача 8* - 4 балла
- Задача 9* - 10 баллов

In [1041]:
```
import numpy as np
import scipy.stats as sps
import matplotlib.pyplot as plt
from statsmodels.distributions.empirical_distribution import ECDF
import pandas as pd
import seaborn as sns
import scipy as sp
from sklearn.datasets import load_boston
from sklearn.metrics import mean_squared_error
from sklearn.cross_validation import train_test_split
from sklearn import linear_model

%matplotlib inline
```

1. Линейная регрессия

Задача 1. По шаблону напишите класс, реализующий линейную регрессию. Интерфейс этого класса в некоторой степени соответствует классу [`LinearRegression` (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression)](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression) из библиотеки `sklearn`.

Модель обучается по формуле

$$\theta = (X^T X)^{-1} X^T Y$$

Несмещенная оценка для параметра σ^2 есть:

$$\widehat{\sigma}^2 = \frac{1}{n-k}\|Y - X\theta\|^2$$

Доверительный интервал уровня доверия α для θ_i есть:

$$\left(\widehat{\theta_i} - \sqrt{a_{i,i}\widehat{\sigma}^2}u_{\frac{1+\alpha}{2}}, \widehat{\theta_i} - \sqrt{a_{i,i}\widehat{\sigma}^2}u_{\frac{1-\alpha}{2}}\right),$$

где $a_{i,i}$ -- диагональный элемент матрицы $(X^T X)^{-1}$, а $u_{\frac{1-\alpha}{2}}$ и $u_{\frac{1+\alpha}{2}}$ соответственно $\frac{1-\alpha}{2}$ - и $\frac{1+\alpha}{2}$ - квантили распределения T_{n-k}

Отклик на новом объекте X вычисляется по формуле

$$Y = X\theta,$$

где θ -- уже обученный вектор весов модели

In [1042]:
```
class LinearRegression:
    def __init__(self):
        super()

    def fit(self, X, Y, alpha=0.95):
        ''' Обучение модели. Предполагается модель Y = X * theta + epsilon,
            где X --- регрессор, Y --- отклик,
            а epsilon имеет нормальное распределение с параметрами (0, sigma^2 * I_n).
            alpha --- уровень доверия для доверительного интервала.
        '''

        self.n, self.k = X.shape

        self.theta = sp.linalg.inv(X.T @ X) @ X.T @ Y
        self.sigma_sq = 1 / (self.n - self.k) * np.sum((Y - X @ self.theta) ** 2)
        help_matrix = np.linalg.inv(X.T @ X)
        shift = np.tile([sps.t(df=self.n - self.k).ppf((1 + alpha) / 2),
                         sps.t(df=self.n - self.k).ppf((1 - alpha) / 2)], (self.k, 1))
        shift = np.tile(np.sqrt(self.sigma_sq * np.diag(help_matrix)), (2, 1)).T * shift
        self.conf_int = np.tile(self.theta, (2, 1)).T - shift

        return self

    def summary(self):
        print('Linear regression on %d features and %d examples' % (self.k, self.n))
        print('Sigma: %.6f' % self.sigma_sq)
        print('\t\tLower\t\tEstimation\tUpper')
        for j in range(self.k):
            print('theta_%d:\t%.6f\t%.6f\t%.6f' % (j, self.conf_int[j, 0],
                                                 self.theta[j], self.conf_int[j, 1]))

    def predict(self, X):
        ''' Возвращает предсказание отклика на новых объектах X. '''

        Y_pred = X @ self.theta
        return Y_pred
```

Загрузите данные о потреблении мороженного в зависимости от температуры воздуха и цены (файл `ice_cream.txt`). Примените реализованный выше класс линейной регрессии к этим данным предполагая, что модель имеет вид $ic = \theta_1 + \theta_2\ t$, где t --- температура воздуха (столбец `temp`), ic --- постребление мороженного в литрах на человека (столбец `IC`). Значения температуры предварительно переведите из Фаренгейта в Цельсий [(Фаренгейт — 32) / 1,8 = Цельсий].

К обученной модели примените фунцию `summary` и постройте график регрессии, то есть график прямой $ic = \widehat{\theta}_1 + \widehat{\theta}_2\ t$, где $\widehat{\theta}_1, \widehat{\theta}_2$ --- МНК-оценки коэффициентов. На график нанесите точки выборки. Убедитесь, что построейнный график совпадает с графиком из презентации с первой лекции, правда, с точностью до значений температура (она была неправильно переведена из Фаренгейта в Цельсий).

In [1043]:
```
df = pd.read_csv('ice_cream.txt', delimiter='\t')
df['temp'] = df['temp'].apply(lambda x: (x - 32) / 1.8)
df.head()
```

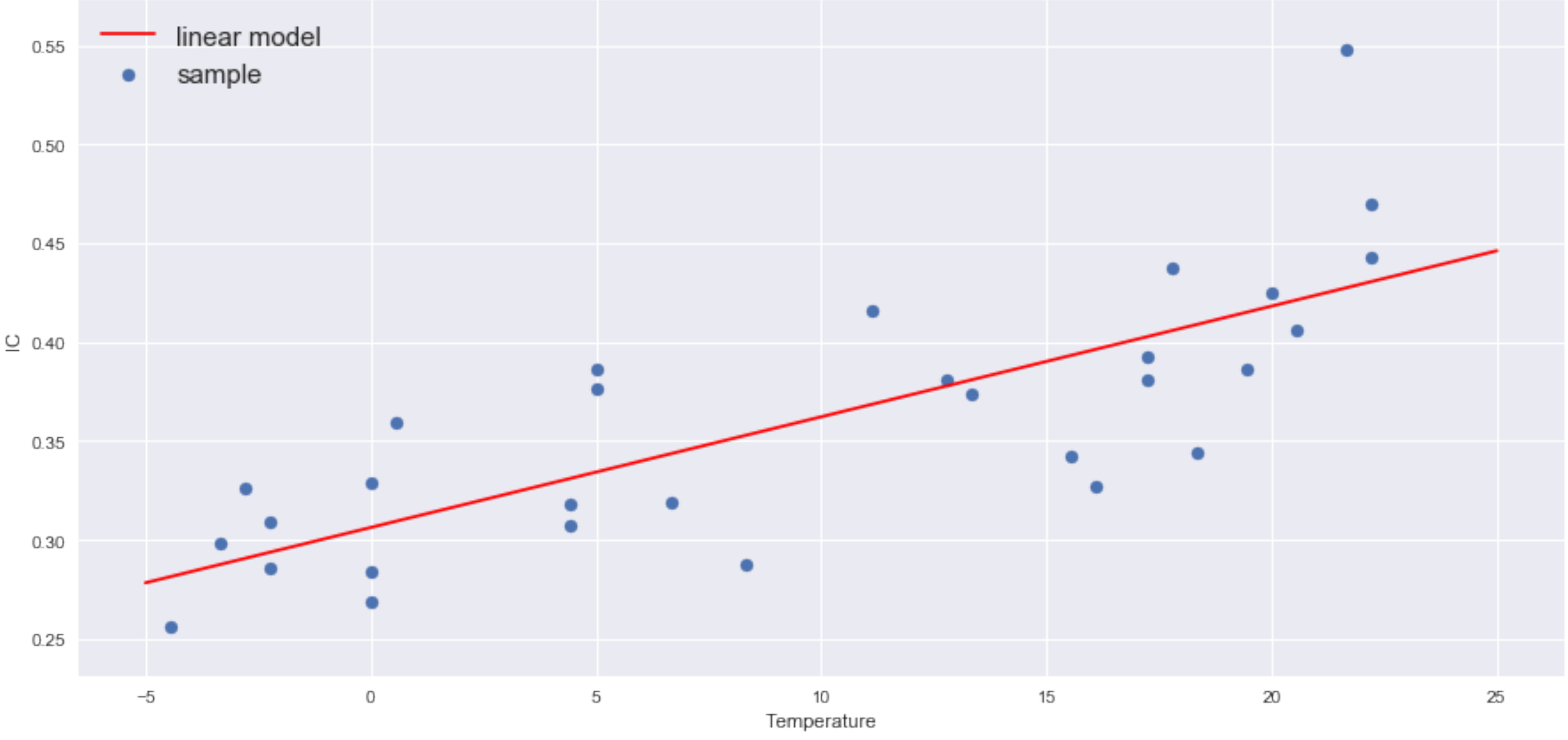
Out[1043]:

	date	IC	price	income	temp	Lag-temp	Year
0	1	0.386	0.270	78	5.000000	56	0
1	2	0.374	0.282	79	13.333333	63	0
2	3	0.393	0.277	81	17.222222	68	0
3	4	0.425	0.280	80	20.000000	69	0
4	5	0.406	0.272	76	20.555556	65	0

In [1044]:
```
my_estimator = LinearRegression()
X = df['temp'].values.reshape(df['temp'].values.shape[0], 1)
X = np.hstack((np.ones_like(X), X))
y = df['IC'].values
my_estimator.fit(X, y)
my_estimator.summary()
```

```
Linear regression on 2 features and 30 examples
Sigma: 0.001786
                Lower           Estimation      Upper
theta_0:        0.283276        0.306298        0.329319
theta_1:        0.003831        0.005593        0.007355
```

In [1045]:
```
plt.figure(figsize=(15, 7))
grid = np.linspace(-5, 25, 1000)
plt.scatter(df['temp'], df['IC'], label='sample')
plt.plot(grid, my_estimator.theta[0] + my_estimator.theta[1] * grid, color='red', label='linear model')
plt.legend(loc='best', fontsize=15)
plt.xlabel('Temperature')
plt.ylabel('IC')
plt.show()
```



Видим, что линейная модель неплохо описывает данные, следовательно, один из тестов корректности нашей линейной регрессии пройден успешно!

Действительно, мой график совпадает с графиком из первой презентации с точностью до перевода температуры в градусы Цельсия:)

Теперь учтите влияние года (столбец `Year`) для двух случаев:

- модель $ic = \theta_1 + \theta_2\ t + \theta_3 y_1 + \theta_4 y_2$, где $y_1 = I\{1\ \text{год}\}, y_2 = I\{2\ \text{год}\}$. Поясните, почему нельзя рассмативать одну переменную y --- номер года.
- для каждого года рассматривается своя линейная зависимость $ic = \theta_1 + \theta_2\ t$.

В каждом случае нарисуйте графики. Отличаются ли полученные результаты? От чего это зависит? Как зависит потребление мороженного от года?

Поясните, почему нельзя рассмативать одну переменную y --- номер года.

Нельзя так делать, потому что омер года -- не числовой признак, а категориальный, а линеные модели ищут закономерности только в числовых данных

Первый случай:

модель $ic = \theta_1 + \theta_2\ t + \theta_3 y_1 + \theta_4 y_2$, где $y_1 = I\{1\ \text{год}\}, y_2 = I\{2\ \text{год}\}$.

In [1046]:
```
df['indicator_1st_year'] = (list(map(lambda x: int(x), df['Year'] == 1)))
df['indicator_2nd_year'] = (list(map(lambda x: int(x), df['Year'] == 2)))
my_estimator = LinearRegression()
X = df[['temp', 'indicator_1st_year', 'indicator_2nd_year']].values
X = np.hstack((np.ones(shape=(X.shape[0], 1)), X))
y = df['IC'].values
my_estimator.fit(X, y)
my_estimator.summary()
```

```
Linear regression on 4 features and 30 examples
Sigma: 0.001016
                Lower           Estimation      Upper
theta_0:        0.251176        0.277050        0.302923
theta_1:        0.004741        0.006095        0.007449
theta_2:        -0.011237       0.016491        0.044218
theta_3:        0.041535        0.074307        0.107078
```

Второй случай:

для каждого года рассматривается своя линейная зависимость $ic = \theta_1 + \theta_2\ t$.

In [1047]:
```
est_0_year = LinearRegression()
est_1_year = LinearRegression()
est_2_year = LinearRegression()
for est, year in zip([est_0_year, est_1_year, est_2_year], [0, 1, 2]):
    X = df[df['Year'] == year]
    X = X['temp'].values.reshape(X['temp'].values.shape[0], 1)
    X = np.hstack((np.ones_like(X), X))
    y = df[df['Year'] == year].IC.values
    est.fit(X, y)
    print('estimator for', year, 'year')
    est.summary()
    print('========================================================')
```

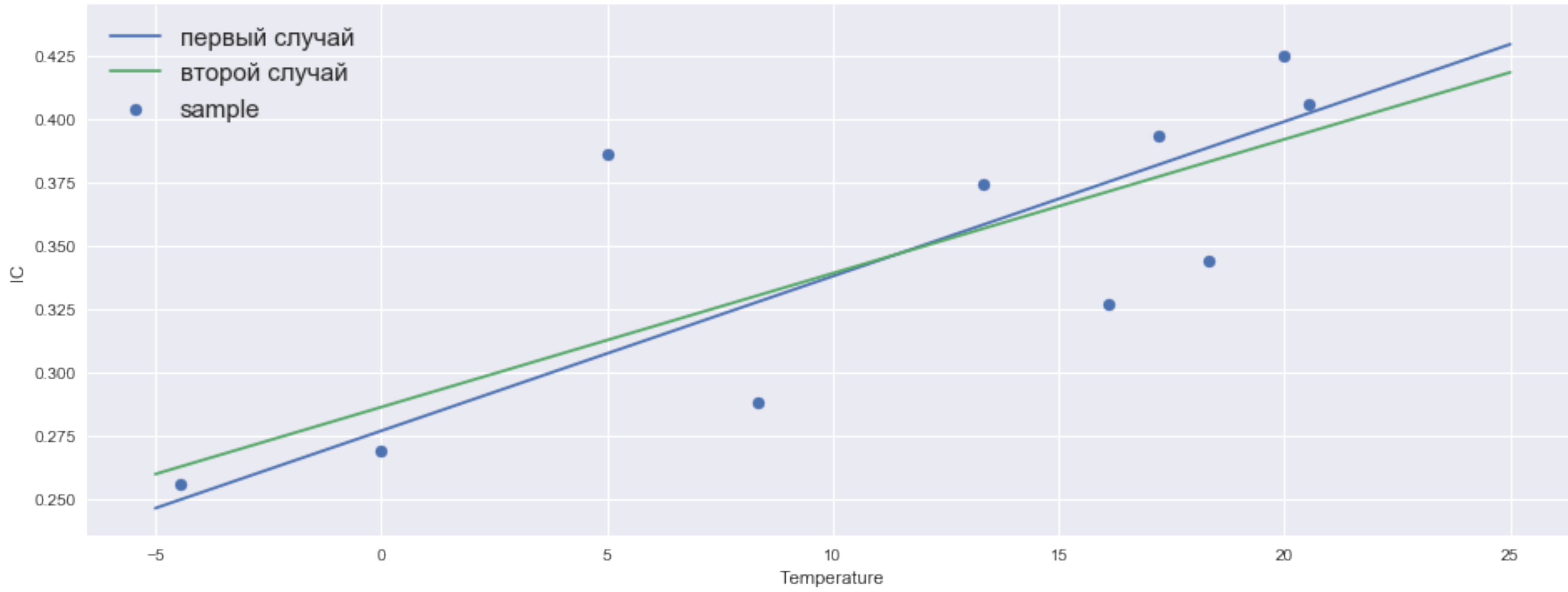
```
estimator for 0 year
Linear regression on 2 features and 10 examples
Sigma: 0.001597
                Lower           Estimation      Upper
theta_0:        0.236963        0.286405        0.335846
theta_1:        0.001787        0.005277        0.008767
========================================================
estimator for 1 year
Linear regression on 2 features and 13 examples
Sigma: 0.000667
                Lower           Estimation      Upper
theta_0:        0.274993        0.297426        0.319859
theta_1:        0.003935        0.005672        0.007409
========================================================
estimator for 2 year
Linear regression on 2 features and 7 examples
Sigma: 0.000766
                Lower           Estimation      Upper
theta_0:        0.303805        0.338346        0.372886
theta_1:        0.004907        0.007877        0.010846
========================================================
```

Построим 3 графика: для каждого года по-отдельности. На каждом графике нарисуем точки выборки и наши линейные модели

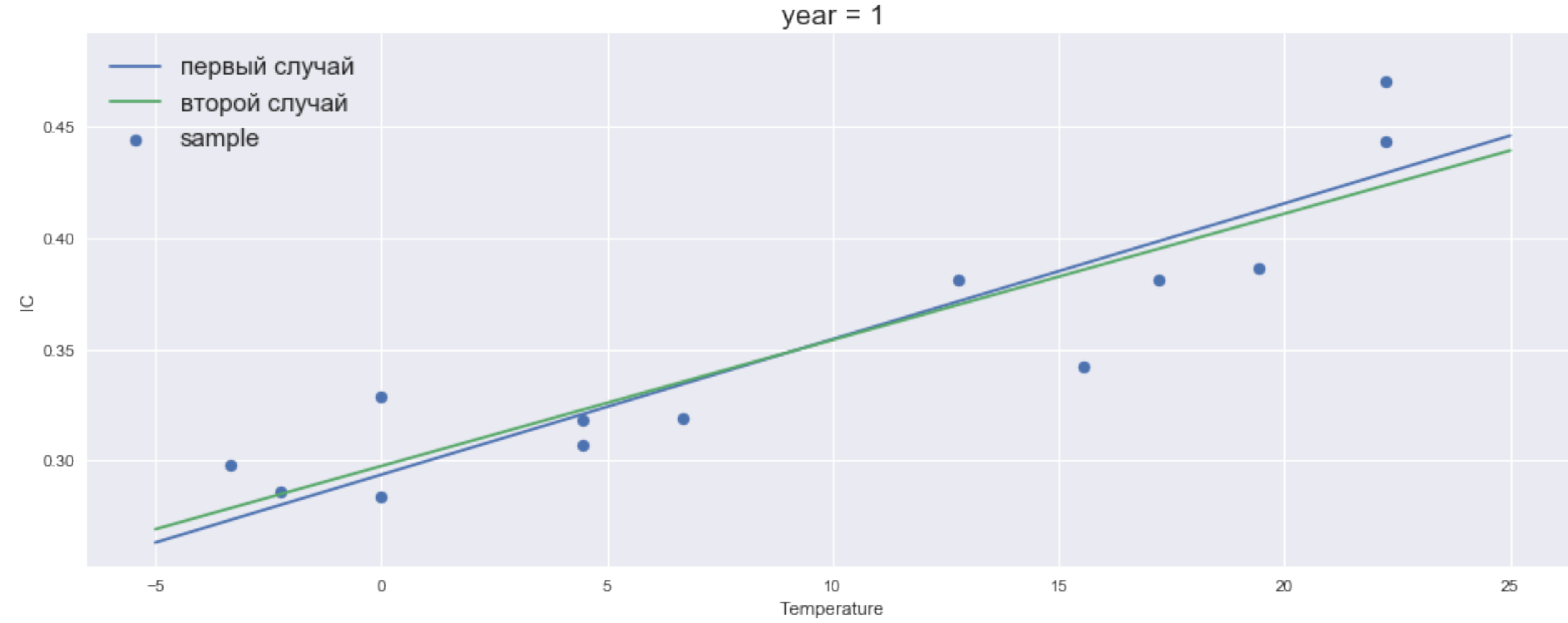
In [1048]:
```
plt.figure(figsize=(16,20))
grid = np.linspace(-5, 25, 1000)
for i, year, second_est in zip([1, 2, 3], [0, 1, 2], [est_0_year, est_1_year, est_2_year]):
    plt.subplot(3, 1, i)
    sample = df[df['Year'] == year].temp
    y = df[df['Year'] == year].IC
    plt.scatter(sample, y, label='sample')
    thetas1 = my_estimator.theta
    print(thetas1)
    plt.plot(grid, thetas1[0] + thetas1[1]*grid + thetas1[2] * (year == 1) + thetas1[3] * (year == 2),
            label='первый случай')
    thetas2 = second_est.theta
    print(thetas2)
    plt.plot(grid, thetas2[0] + thetas2[1] * grid, label='второй случай')
    plt.legend(loc='best', fontsize=15)
    plt.xlabel('Temperature')
    plt.ylabel('IC')
    plt.title('year = {}'.format(year), fontsize=17)
plt.show()
```

```
[ 0.27704976  0.00609468  0.01649056  0.07430653]
[ 0.28640471  0.00527726]
[ 0.27704976  0.00609468  0.01649056  0.07430653]
[ 0.29742583  0.00567179]
[ 0.27704976  0.00609468  0.01649056  0.07430653]
[ 0.33834553  0.00787659]
```

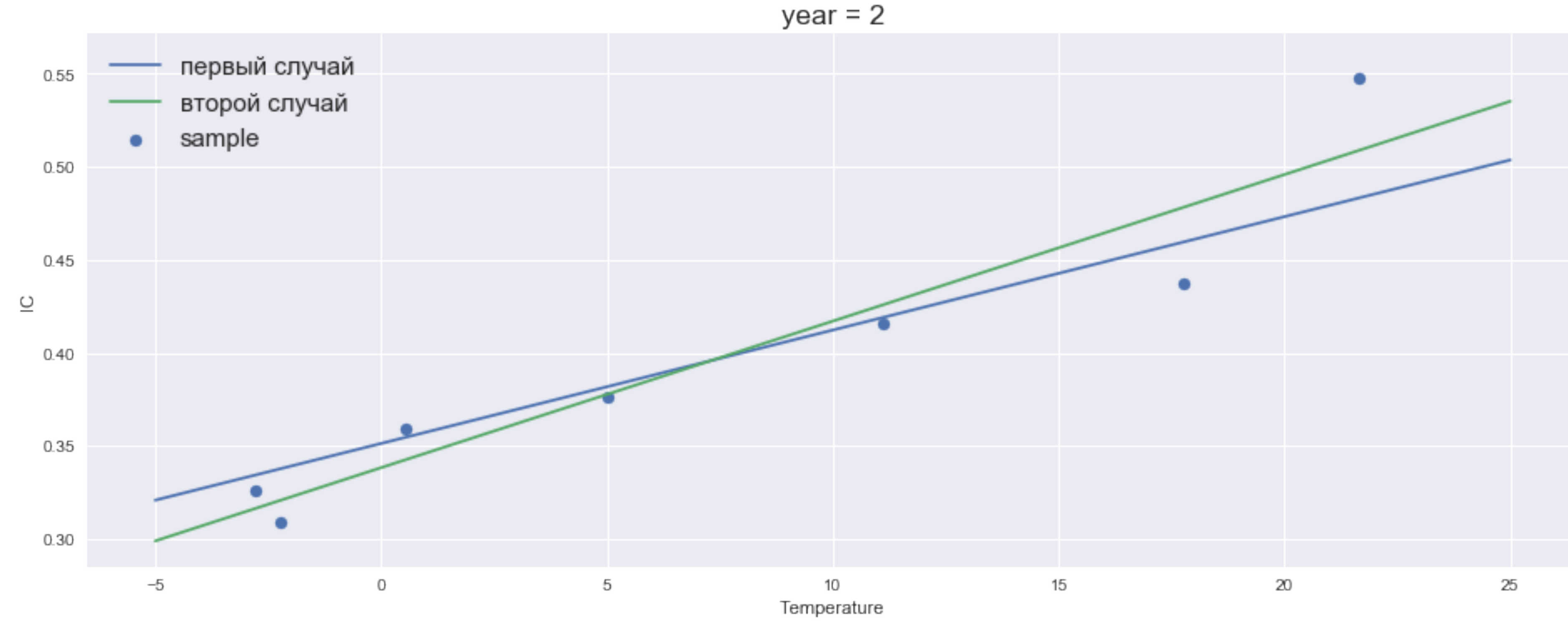
year = 0



year = 1

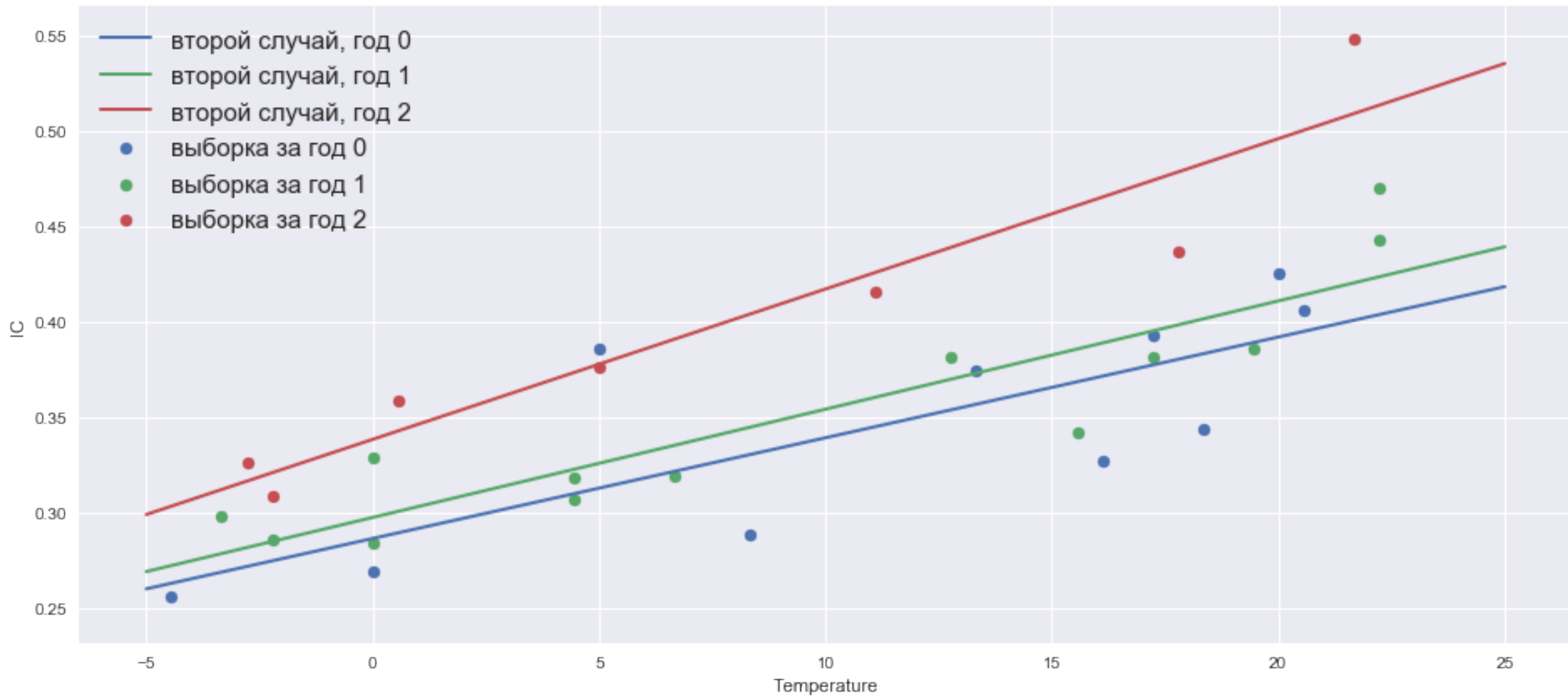


year = 2



Или все это на одном графике

In [1049]:
```
plt.figure(figsize=(16,7))
grid = np.linspace(-5, 25, 1000)
for year, second_est in zip([0, 1, 2], [est_0_year, est_1_year, est_2_year]):
    sample = df[df['Year'] == year].temp
    y = df[df['Year'] == year].IC
    plt.scatter(sample, y, label='выборка за год {}'.format(year))
    thetas2 = second_est.theta
    plt.plot(grid, thetas2[0] + thetas2[1] * grid,
             label='второй случай, год {}'.format(year), linewidth=2)
plt.legend(loc='best', fontsize=15)
plt.xlabel('Temperature')
plt.ylabel('IC')
plt.show()
```



Графики чуточку разнятся из-за того, что в первом случае мы обучаемся на всей выборке, а во втором случае мы обучаем линейную регрессию под каждый год отдельно, без учета других лет Из графиков можем сделать вывод, что со временем (при увеличении года) количество потребляемого мороженого растет

Наконец, обучите модель на предсказание потребления мороженного в зависимости от всех переменных. Не забудьте, что для года нужно ввести две переменных. Для полученной модели выведите `summary`.

In [1050]:
```
X = df.drop(['date', 'Year', 'IC'], axis=1).values
y = df['IC'].values
est = LinearRegression()
est.fit(X, y)
est.summary()
```

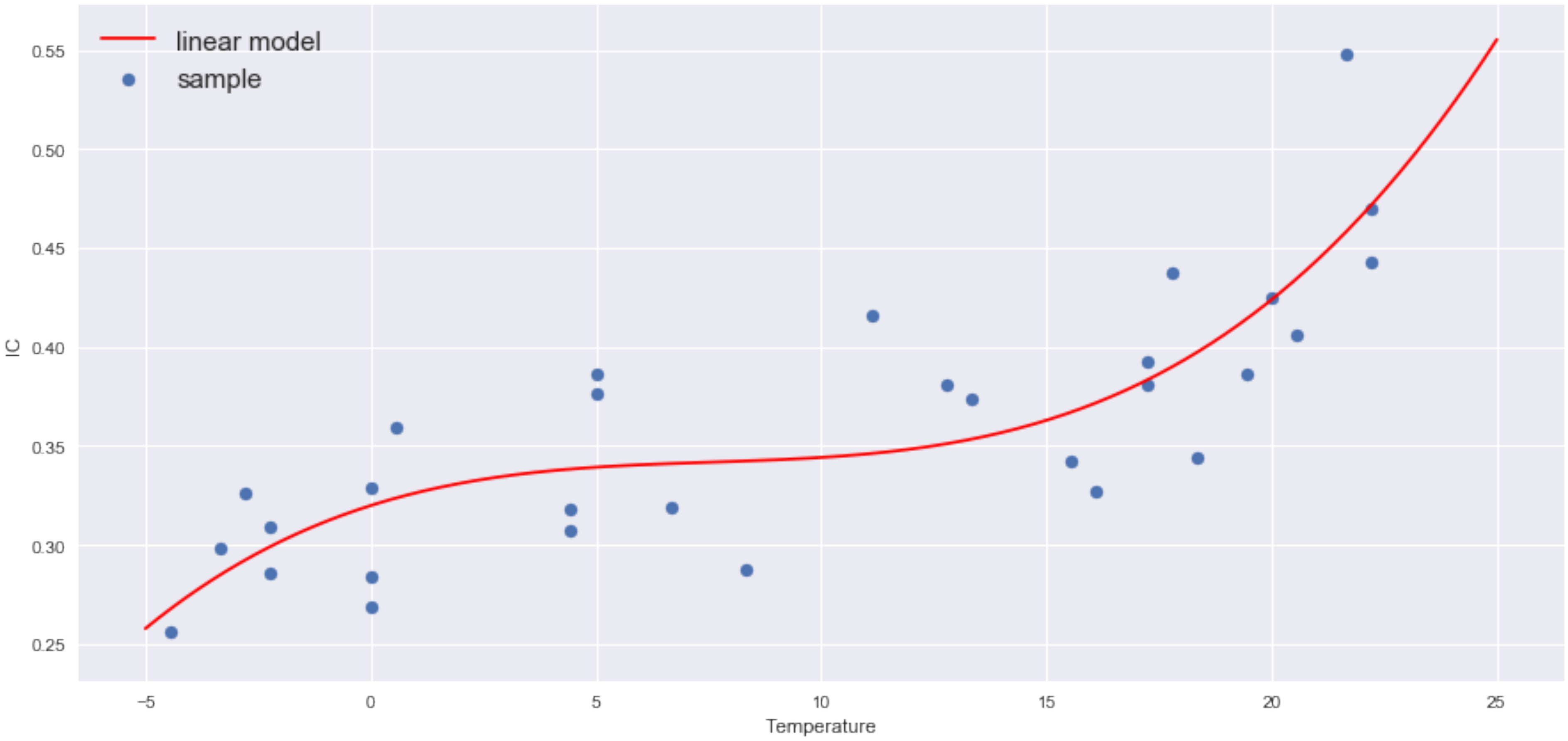
```
Linear regression on 6 features and 30 examples
Sigma: 0.001233
                Lower           Estimation      Upper
theta_0:        -0.145322       1.042379        2.230081
theta_1:        -0.004028       -0.000038       0.003951
theta_2:        0.004574        0.006460        0.008346
theta_3:        -0.001113       -0.000180       0.000753
theta_4:        -0.023702       0.010091        0.043884
theta_5:        0.007848        0.078241        0.148635
```

Но это еще не все. Постройте теперь линейную регрессию для модели $ic = \theta_1 + \theta_2\, t + \theta_3\, t^2 + \theta_4\, t^3$. Выведите для нее summary и постройте график предсказания, то есть график кривой $ic = \widehat{\theta}_1 + \widehat{\theta}_2\, t + \widehat{\theta}_3\, t^2 + \widehat{\theta}_4\, t^3$. Хорошие ли получаются результаты?

In [1051]:
```
X = df['temp'].values
X = X.reshape(X.shape[0], 1)
X = np.hstack((X**0, X ** 1, X**2, X**3))
est = LinearRegression()
est.fit(X, y)
est.summary()
```

```
Linear regression on 4 features and 30 examples
Sigma: 0.001529
                Lower           Estimation      Upper
theta_0:        0.295294        0.319902        0.344510
theta_1:        0.000388        0.007200        0.014013
theta_2:        -0.001861       -0.000855       0.000152
theta_3:        0.000002        0.000038        0.000073
```

In [1052]:
```
plt.figure(figsize=(15, 7))
grid = np.linspace(-5, 25, 1000)
plt.scatter(df['temp'], df['IC'], label='sample')
thetas = est.theta
plt.plot(grid, thetas[0] + thetas[1] * grid + thetas[2] * grid **2 + thetas[3] * grid ** 3,
         color='red', label='linear model')
plt.legend(loc='best', fontsize=15)
plt.xlabel('Temperature')
plt.ylabel('IC')
plt.show()
```



Действительно, на глаз получилось заметно лучше, но как мне кажется, это похоже на переобучение, на что также намекают маленькие значения θ

Чтобы понять, почему так происходит, выведите значения матрицы $(X^TX)^{-1}$ для данной матрицы и посчитайте для нее индекс обусловленности $\sqrt{\lambda_{max}/\lambda_{min}}$, где $\lambda_{max}, \lambda_{min}$ --- максимальный и минимальный собственные значения матрицы X^TX. Собственные значения можно посчитать функцией `scipy.linalg.eigvals` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigvals.html).

Прокомментируйте полученные результаты. Помочь в этом может следующая статья (https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%BE%D0%B1%D1%83%D1%81%D0%BB%D0%BE%D0%

In [1053]:
```
def condition_number(X):
    eigvals = np.linalg.eigvals(np.linalg.inv(X.T @ X))
    return np.sqrt(max(eigvals) / min(eigvals))
```

In [1054]:
```
condition_number(X)
```

Out[1054]: 8140.3947488976664

Получили очень большой индекс обусловленности, что говорит нам о мультиколлинеарности, которая в свою очередь ведет к переобучению (мы уже можем говорить о мультиколлинеарности когда индекс обусловленности больше 30)

Выводы подводил по ходу дела. В целом затестили наш алгоритм линейной регрессии, работает стабильно:)

Задача 2. В данной задаче нужно реализовать функцию отбора признаков для линейной регрессии. Иначе говоря, пусть есть модель $y = \theta_1 x_1 + \ldots + \theta_k x_k$. Нужно определить, какие θ_j нужно положить равными нулю, чтобы качество полученной модели было максимальным.

Для этого имеющиеся данные нужно случайно разделить на две части --- обучение и тест (train и test). На первой части нужно обучить модель регресии, взяв некоторые из признаков, то есть рассмотреть модель $y = \theta_{j_1} x_{j_1} + \ldots + \theta_{j_s} x_{j_s}$. По второй части нужно посчитать ее качество --- среднеквадратичное отклонение (mean squared error) предсказания от истинного значения отклика, то есть величину

$$MSE = \sum_{i \in test} \left(\widehat{y}(x_i) - Y_i\right)^2,$$

где $x_i = (x_{i,1}, \ldots, x_{i,k})$, Y_i --- отклик на объекте x_i, а $\widehat{y}(x)$ --- оценка отклика на объекте x.

Если k невелико, то подобным образом можно перебрать все поднаборы признаков и выбрать наилучший по значению MSE.

Для выполнения задания воспользуйтесь следующими функциями:

- `sklearn.linear_model.LinearRegression` (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression) --- реализация линейной регрессии. В данной реализации свободный параметр θ_1 по умолчанию автоматически включается в модель. Отключить это можно с помощью `fit_intercept=False`, но это не нужно. *В данной задаче требуется, чтобы вы воспользовались готовой реализацией линейной регрессии, а не своей. Ведь на практике важно уметь применять готовые реализации, а не писать их самостоятельно.*
- `sklearn.cross_validation.train_test_split` (http://scikit-learn.org/0.16/modules/generated/sklearn.cross_validation.train_test_split.html) --- функция разбиения данных на train и test. Установите параметр `test_size=0.3`.
- `sklearn.metrics.mean_squared_error` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html) --- реализация MSE.

Для перебора реализуйте функцию.

In [1055]:
```
from sklearn.linear_model import LinearRegression
```

In [1056]:
```
def best_features(X_train, X_test, Y_train, Y_test):
    mses = []  # сюда записывайте значения MSE
    k = X_train.shape[1]

    for j in range(1, 2 ** k):  # номер набора признаков
        mask = np.array([j & (1 << s) for s in range(k)], dtype=bool)
        features_numbers = np.arange(k)[mask]  # набор признаков
        est = LinearRegression()
        est.fit(X_train[:, features_numbers], Y_train)
        Y_pred = est.predict(X_test[:, features_numbers])
        mse = mean_squared_error(Y_test, Y_pred)  # MSE для данного набора признаков
        mses.append(mse)

    # Печать 10 лучших наборов
    print('mse\t features')
    mses = np.array(mses)
    best_numbres = np.argsort(mses)[:10]
    for j in best_numbres:
        mask = np.array([j & (1 << s) for s in range(k)], dtype=bool)
        features_numbers = np.arange(k)[mask]
        print('%.3f\t' % mses[j], features_numbers)
```

Примените реализованный отбор признаков к датасетам

- [Yacht Hydrodynamics (http://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics)](http://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics) --- для парусных яхт нужно оценить остаточное сопротивление на единицу массы смещения (последний столбец) в зависимости от различных характеристик яхты.
- [Boston Housing Prices (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston)](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston) --- цены на дома в Бостоне в зависимости от ряда особенностей.

In [1057]:
```
!wget 'http://archive.ics.uci.edu/ml/machine-learning-databases/00243/yacht_hydrodynamics.data'
```

```
--2017-05-21 12:35:51--  http://archive.ics.uci.edu/ml/machine-learning-databases/00243/yacht_hydrodynamic
s.data (http://archive.ics.uci.edu/ml/machine-learning-databases/00243/yacht_hydrodynamics.data)
Resolving archive.ics.uci.edu... 128.195.10.249
Connecting to archive.ics.uci.edu|128.195.10.249|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11487 (11K) [text/plain]
Saving to: ���yacht_hydrodynamics.data.5���

yacht_hydrodynamics 100%[===================>]  11.22K  --.-KB/s    in 0s

2017-05-21 12:35:51 (80.6 MB/s) - ���yacht_hydrodynamics.data.5��� saved [11487/11487]
```

In [1059]:
```
names = ['Longitudinal position of the center of buoyancy', 'Prismatic coefficient',
 'Length-displacement ratio', 'Beam-draught ratio', 'Length-beam ratio',
 'Froude number', 'Residuary resistance per unit weight of displacement']
yacht_df = pd.read_csv('yacht_hydrodynamics.data.1', header=None, names=names, delim_whitespace=True)
boston = load_boston()
```

Yacht Hydrodynamics

In [1060]:
```
yacht_df.head()
```

Out[1060]:

	Longitudinal position of the center of buoyancy	Prismatic coefficient	Length-displacement ratio	Beam-draught ratio	Length-beam ratio	Froude number	Residuary resistance per unit weight of displacement
0	-2.3	0.568	4.78	3.99	3.17	0.125	0.11
1	-2.3	0.568	4.78	3.99	3.17	0.150	0.27
2	-2.3	0.568	4.78	3.99	3.17	0.175	0.47
3	-2.3	0.568	4.78	3.99	3.17	0.200	0.78
4	-2.3	0.568	4.78	3.99	3.17	0.225	1.18

In [1061]:
```
X_train, X_test, Y_train, Y_test = train_test_split(yacht_df.values[:, :6],
                                                    yacht_df.values[:, 6],
                                                    test_size=0.3, random_state=42)
best_features(X_train, X_test, Y_train, Y_test)
```

```
mse      features
77.584   [1 4 5]
77.592   [0 4 5]
77.597   [1 5]
77.604   [0 5]
77.616   [0 1 2 5]
77.622   [0 1 2 4 5]
77.632   [3 5]
77.635   [0 1 2 3 4]
77.636   [3 4 5]
77.639   [0 1 2 3 5]
```

In [1062]:
```
print(len(Y_test), Y_test.mean())
```

```
93 8.23827956989
```

Видим, что самым мощным признаком является признак номер 5, соответствующий числу Фруда ([https://ru.wikipedia.org/wiki/Число_Фруда (https://ru.wikipedia.org/wiki/Число_Фруда)](https://ru.wikipedia.org/wiki/Число_Фруда)), все остальные чередуются. Посмотрев на среднее значение по тесту можно сказать, что лучшая модель ошибается в среднем на 0.9.

Boston Housing Prices

In [1063]:
```
X_train, X_test, Y_train, Y_test = train_test_split(boston.data,
                                                    boston.target,
                                                    test_size=0.3, random_state=42)
best_features(X_train, X_test, Y_train, Y_test)
```

```
mse      features
20.762   [ 0  4  5  7  8  9 10 12]
20.786   [ 1  4  5  7  8  9 10 12]
20.801   [ 0  2  4  5  7  8  9 10 12]
20.862   [ 1  2  4  5  7  8  9 10 12]
20.886   [ 1  3  4  5  7  8  9 10 12]
20.909   [ 0  3  4  5  7  8  9 10 11 12]
20.910   [ 0  4  5  7  8  9 10 11 12]
20.918   [ 1  4  5  6  7  8  9 10 12]
20.925   [ 1  2  3  4  5  7  8  9 10 12]
20.954   [ 0  2  3  4  5  7  8  9 10 11 12]
```

In [1065]:
```
print(len(Y_test), Y_test.mean())
```

```
152 21.4078947368
```

Тут мы видим ситауцию "каждая фича вносит свой маленький вклад, а если их объединить, то получается хорошая модель". В этом датасете наша линейная модель может уже достаточно хорошо восстанавливать целевую переменную.

Задача 3*. Загрузите датасет (http://people.sc.fsu.edu/~jburkardt/datasets/regression/x01.txt), в котором показана зависимость веса мозга от веса туловища для некоторых видов млекопитающих. Задача состоит в том, чтобы подобрать по этим данным хорошую модель регрессии. Для этого, можно попробовать взять некоторые функции от значения веса туловища, например, степенную, показательную, логарифмическую. Можно также сделать преобразование значений веса мозга, например, прологарифмировать. Кроме того, можно разбить значения веса туловища на несколько частей и на каждой части строить свою модель линейной регрессии.

In [1066]:
```
!wget http://people.sc.fsu.edu/~jburkardt/datasets/regression/x01.txt
```

```
--2017-05-21 12:37:07--  http://people.sc.fsu.edu/~jburkardt/datasets/regression/x01.txt
(http://people.sc.fsu.edu/~jburkardt/datasets/regression/x01.txt)
Resolving people.sc.fsu.edu... 144.174.16.100
Connecting to people.sc.fsu.edu|144.174.16.100|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2048 (2.0K) [text/plain]
Saving to: ���x01.txt.1���

x01.txt.1           100%[===================>]   2.00K  --.-KB/s    in 0s

2017-05-21 12:37:07 (122 MB/s) - ���x01.txt.1��� saved [2048/2048]
```

In [1067]:
```
f = open('x01.txt')
data = []
for line in f.readlines():
    if line[0] != '#':
        data.append(line)
```

In [1068]:
```
columns = [s.strip() for s in data[2:5]]
columns[1:]
```

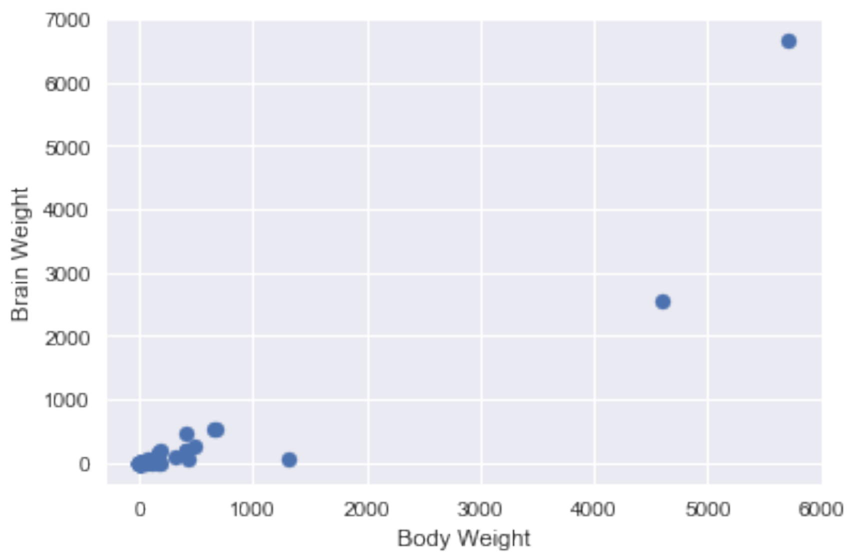
Out[1068]:
```
['Brain Weight', 'Body Weight']
```

In [1069]:
```
dat = np.array(list(map(lambda s: [np.float(s_i) for s_i in s.split()][1:], data[5:-1])))
df = pd.DataFrame(dat, columns=columns[1:])
df.head()
```

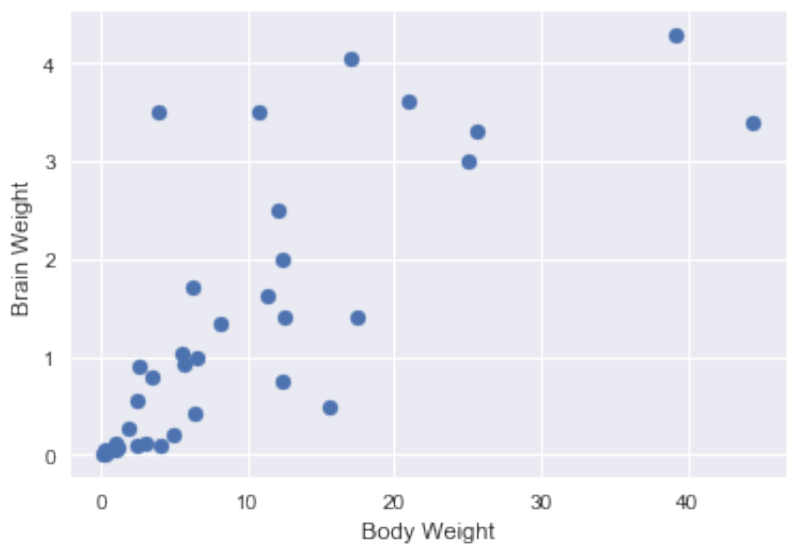
Out[1069]:

	Brain Weight	Body Weight
0	3.385	44.5
1	0.480	15.5
2	1.350	8.1
3	465.000	423.0
4	36.330	119.5

In [1070]:
```
plt.scatter(df['Body Weight'], df['Brain Weight'])
plt.xlabel('Body Weight')
plt.ylabel('Brain Weight')
plt.show()
```

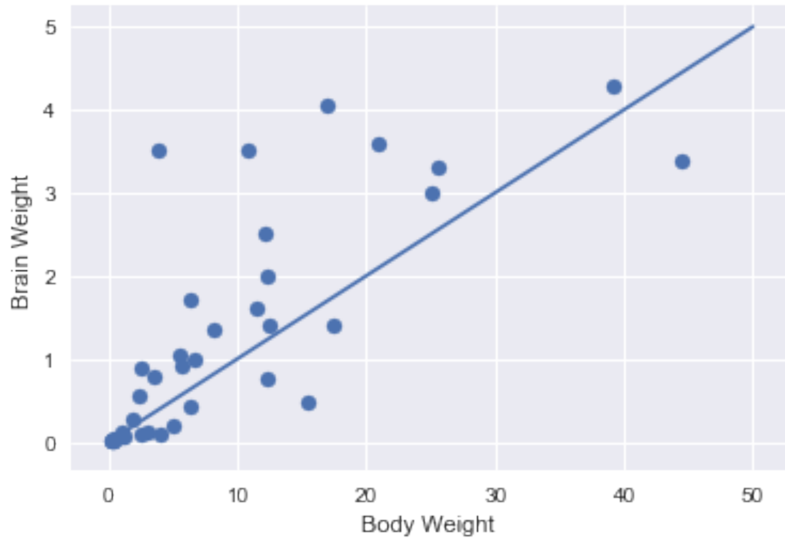


In [1073]:
```
df_small = df[df['Body Weight'] < 50]
plt.scatter(df_small['Body Weight'], df_small['Brain Weight'])
plt.xlabel('Body Weight')
plt.ylabel('Brain Weight')
plt.show()
```

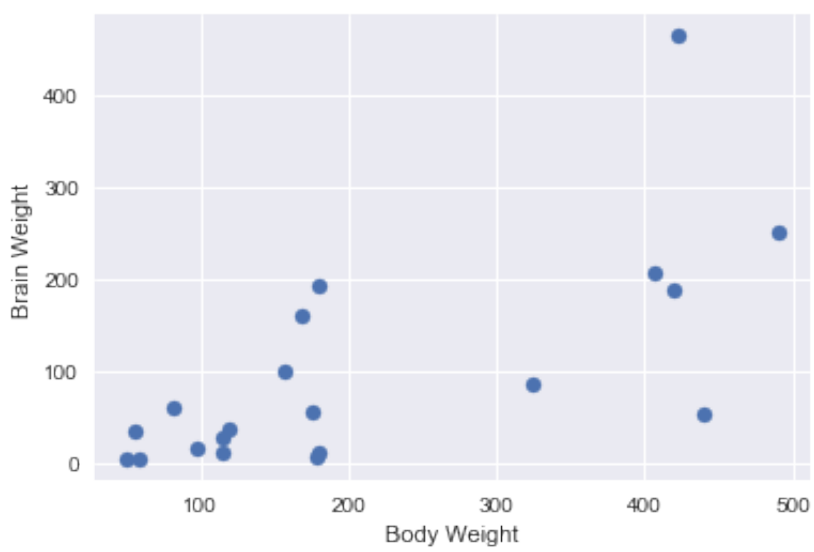


In [1075]:
```
small_estimator = LinearRegression(fit_intercept=True)
X, y = df_small['Body Weight'].values.reshape(-1, 1), df_small['Brain Weight'].values.reshape(-1, 1)
X_ones = np.hstack((np.ones_like(X), X))
small_estimator.fit(X_ones, y)
print(small_estimator.score(X_ones, y))
print(small_estimator.coef_)
theta = small_estimator.coef_[0]
grid = np.linspace(0, 50, 1000)
plt.plot(grid, theta[0] + theta[1]* grid)
plt.scatter(df_small['Body Weight'], df_small['Brain Weight'])
plt.xlabel('Body Weight')
plt.ylabel('Brain Weight')
plt.show()
```

```
0.594911175385
[[ 0.          0.09991637]]
```

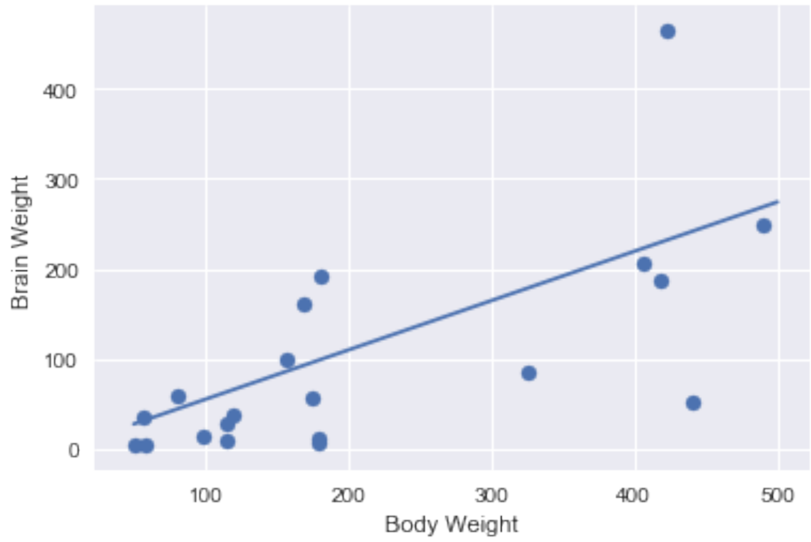


In [1086]:
```
df_med = df[50 <= df['Body Weight']]
df_med = df_med[df_med['Body Weight'] < 500]
plt.scatter(df_med['Body Weight'], df_med['Brain Weight'])
plt.xlabel('Body Weight')
plt.ylabel('Brain Weight')
plt.show()
```

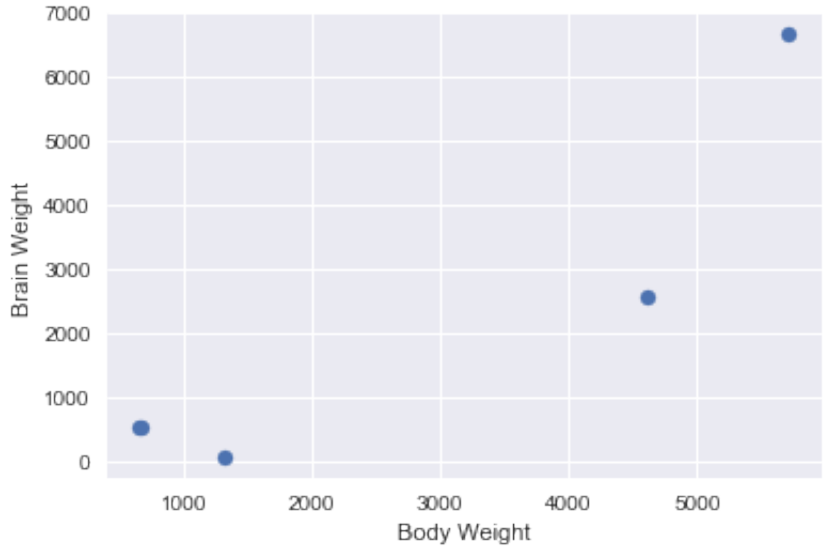


In [1087]:
```
med_estimator = LinearRegression(fit_intercept=True)
X, y = df_med['Body Weight'].values.reshape(-1, 1), df_med['Brain Weight'].values.reshape(-1, 1)
X_ones = np.hstack((np.ones_like(X), X))
med_estimator.fit(X_ones, y)
print(med_estimator.score(X_ones, y))
print(med_estimator.coef_)
theta = med_estimator.coef_[0]
grid = np.linspace(50, 500, 1000)
plt.plot(grid, theta[0] + theta[1]* grid)
plt.scatter(df_med['Body Weight'], df_med['Brain Weight'])
plt.xlabel('Body Weight')
plt.ylabel('Brain Weight')
plt.show()
```

```
0.478582920648
[[ 0.          0.54872072]]
```

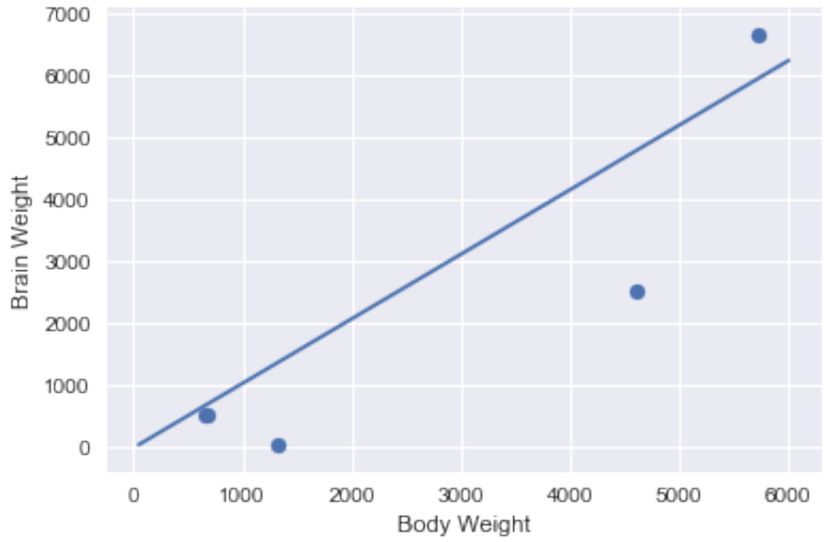


In [1088]:
```
df_large = df[500 <= df['Body Weight']]
plt.scatter(df_large['Body Weight'], df_large['Brain Weight'])
plt.xlabel('Body Weight')
plt.ylabel('Brain Weight')
plt.show()
```



In [1089]:
```
large_estimator = LinearRegression(fit_intercept=True)
X, y = df_large['Body Weight'].values.reshape(-1, 1), df_large['Brain Weight'].values.reshape(-1, 1)
X_ones = np.hstack((np.ones_like(X), X))
large_estimator.fit(X_ones, y)
print(large_estimator.score(X_ones, y))
print(large_estimator.coef_)
theta = large_estimator.coef_[0]
grid = np.linspace(50, 6000, 1000)
plt.plot(grid, theta[0] + theta[1]* grid)
plt.scatter(df_large['Body Weight'], df_large['Brain Weight'])
plt.xlabel('Body Weight')
plt.ylabel('Brain Weight')
plt.show()
```

```
0.824084739615
[[ 0.          1.04201788]]
```



На самом деле обучать линейную регрессию по 4рем точкам -- такое себе, скорее всего, это сразу переобучение, но все же они никуда не вписываются

Вывод: Много комментариев было сделано по ходу дела. Считаю, что построить несколько регрессий и применять их в зависимости от входных данных будет являться наилучшим решением задачи. Заключил это просто взглянув на графики (не забыл о бикини:)).

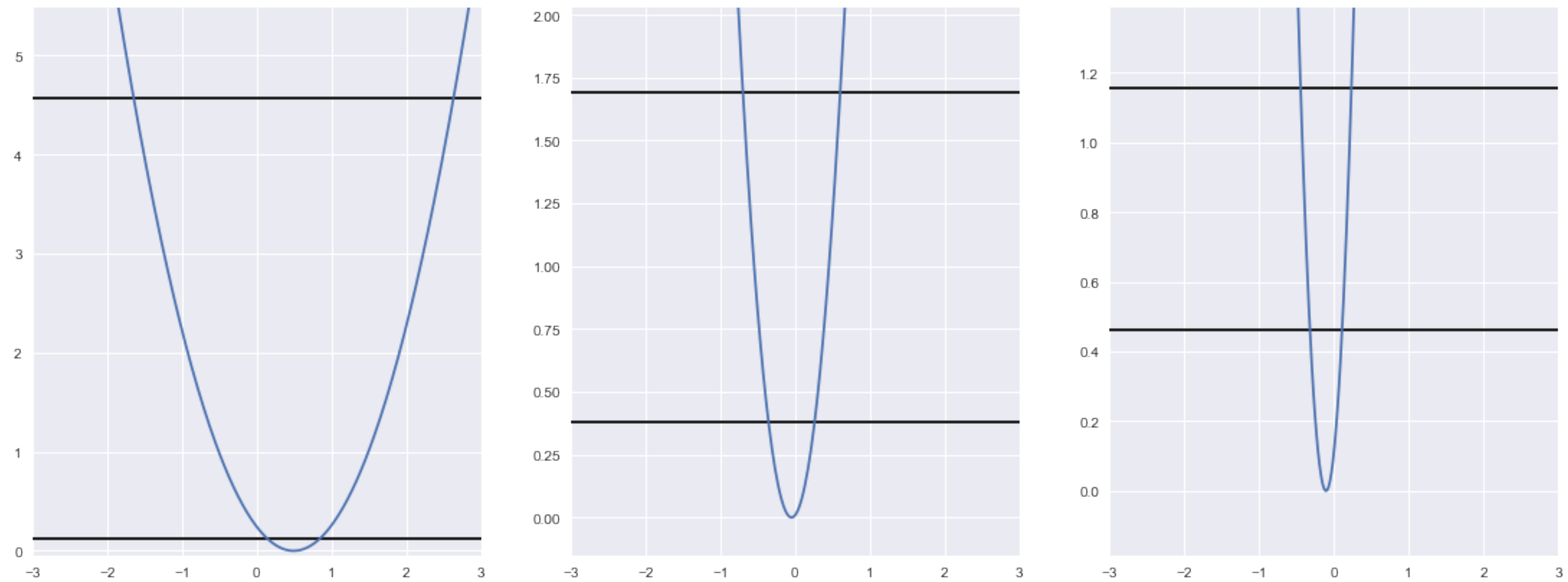
Задача 4. Пусть $X_1, \dots, X_n$ --- выборка из распределения $\mathcal{N}(a, \sigma^2)$. Постройте точную доверительную область для параметра $\theta = (a, \sigma^2)$ уровня доверия $\alpha = 0.95$ для сгенерированной выборки размера $n \in \{5, 20, 50\}$ из стандартного нормального распределения. Какой вывод можно сделать?

Из домашнего задания:

$$\sigma^2 \geq \frac{n(\overline{X} - a)^2}{z^2_{\frac{1+\alpha_1}{2}}} \quad \& \quad \frac{nS^2}{u_{p_1}} \leq \sigma^2 \leq \frac{nS^2}{u_{p_2}}$$

где z_α -- α - квантиль стандартного нормального распределения, u_α -- α - квантиль χ^2_{n-1}, а числа $\alpha_1, \alpha_2, p_1, p_2$ имеют зависимость: $\alpha_1 \alpha_2 = \alpha, p_2 - p_1 = \alpha_2$.

In [1098]:
```
n_array = [5, 20, 50]
alpha = 0.95
alpha_1 = np.sqrt(alpha)
alpha_2 = alpha / alpha_1
plt.figure(figsize=(19, 7))
for n, i in zip(n_array, [1, 2, 3]):
    sample = sps.norm.rvs(size=n)
    plt.subplot(1, 3, i)
    vertical_lines = (n * (np.mean(sample ** 2) - np.mean(sample) ** 2))
    u_p1, u_p2 = sps.chi2.interval(alpha_2, n-1)
    down = vertical_lines / u_p2
    up = vertical_lines / u_p1
    plt.hlines(up, -5, 5)
    plt.hlines(down, -5, 5)
    parab = lambda a: n * (sample.mean() - a) ** 2 / sps.norm.ppf((1 + alpha_1) / 2) ** 2
    grid = np.linspace(-10, 10, 1000)
    plt.plot(grid, list(map(lambda x: parab(x), grid)))
    plt.ylim(-down * 0.4, up * 1.2)
    plt.xlim(-3, 3)
plt.show()
```



Вывод: Доверительная область получилоась достаточно небольшой, учитывая то, что мы не знаем ни среднее, ни медиану распределения. Тем более, с ростом выборки область уменьшается и истинное значение точно попадает в эту область

Задача 5*. Пусть дана линейная гауссовская модель $Y = X\theta + \varepsilon$, где $\varepsilon \sim \mathcal{N}(0, \beta^{-1}I_n)$. Пусть θ имеет априорное распределение $\mathcal{N}(0, \alpha^{-1}I_k)$. Такая постановка задачи соответствует Ridge-регрессии. Оценкой параметров будет математическое ожидание по апостериорному распределению, аналогично можно получить доверительный интервал. Кроме того, с помощью апостериорного распределения можно получить доверительный интервал для отклика на новом объекте, а не только точечную оценку.

Реализуйте класс RidgeRegression подобно классу LinearRegression, но добавьте в него так же возможность получения доверительного интервала для отклика на новом объекте. Примените модель к некоторых датасетам, которые рассматривались в предыдущих задачах. Нарисуйте графики оценки отклика на новом объекте и доверительные интервалы для него.

Ридж-регрессия осуществляет отбор признаков с помощью $L2$-регялризации. Формула вычисления весов алгоритма есть

$\widehat{\theta} = (X^TX + aE)^{-1}X^TY$

Почему это круто?

У матриц X^TX и $X^TX + aE$ собвственные вектора совпадают, а собственные значения отличаются ровно на a:

$$\sqrt{\frac{\lambda_{max} + a}{\lambda_{min} + a}}$$

Получаем, что чем больше число a, тем меньше число обусловленности, поэтому с ростом a возрастает устойчивость задачи

In [900]:
```
class RigeRegression():
    def fit(self, X, y, alpha, beta):
        self.n, self.k = X.shape
        self.alpha = alpha
        self.beta = beta
        self.theta = np.linalg.inv(beta * np.ones(shape=(self.n, self.n)) + X.T @ X) @ X.T @ y

    def predict(X):
        Y_pred = X @ self.theta
        return Y_pred

    def posteriori(X):
        mean = self.theta
        A = np.linalg.inv(beta * np.ones(shape=(self.n, self.n)) + X.T @ X)
        sigma_sqr = self.alpha * A @ X.T @ X @ A
        return sps.norm(loc=mean, scale=np.sqrt(sigma_sqr))
```

Опять обидно за то, что не сделал эту задачу во второй раз (первый был на семинаре). Ну здесь уже просто не хватило времени

2. Проверка статистических гипотез

Задача 6. Существует примета, что если перед вам дорогу перебегает черный кот, то скоро случится неудача. Вы же уже достаточно хорошо знаете статистику и хотите проверить данную примету. Сформулируем задачу на математическом языке. Пусть $X_1, \dots, X_n \sim Bern(p)$ --- проведенные наблюдения, где $X_i = 1$, если в i-м испытании случилась неудача после того, как черный кот перебежал дорогу, а p --- неизвестная вероятность такого события. Нужно проверить гипотезу $H_0 : p = 1/2$ (отсутствие связи между черным котом и неудачей) против альтернативы $H_1 : p > 1/2$ (неудача происходит чаще если черный кот перебегает дорогу).

Известно, что $S = \{T(X) > c_\alpha\}$, где $T(X) = \sum X_i$, является равномерно наиболее мощным критерием для данной задачи. Чему при этом равно c_α? При этом p-value в данной задаче определяется как $p(t) = \mathsf{P}_{0.5}(T(X) > t)$, где $t = \sum x_i$ --- реализация статистики $T(X)$.

Для начала проверьте, что критерий работает. Возьмите несколько значений n и реализаций статистики $T(X)$. В каждом случае найдите значение c_α и p-value. Оформите это в виде таблицы.

Пользуйтесь функциями из `scipy.stats`, про которые подробно написано в файле `python_5`. Внимательно проверьте правильность строгих и нестрогих знаков.

Понятно, что никакой связи между черными кошками, перебегающими дорогу и неудачами нет, поэтому сгенерируем выборку из бернуллиевского с параметром $p = \frac{1}{2}$. $T(X)$ имеет распределение $Bin(n, \theta)$, поэтому понятно, что c_α есть $1 - \alpha$ - квантиль распределения $Bin(n, \theta_0)$

In [1103]:
```
n_array = [5, 10, 20, 30, 50]
alpha = 0.05
statistics = []
T = sps.bernoulli(p=0.5).rvs(size=50)
for n in n_array:
    t = np.sum(T[:n])
    c = sps.binom(n=n, p=0.5).ppf(1 - alpha)
    pvalue = sps.binom(n, 0.5).sf(t)
    statistics.append([t, pvalue, c])
pd.DataFrame(data=statistics, columns=['t', 'p-value', 'c'])
```

Out[1103]:

	t	p-value	c
0	1	0.812500	4.0
1	2	0.945312	8.0
2	6	0.942341	14.0
3	14	0.572232	19.0
4	23	0.664094	31.0

Видим, что на уровне значимости 0.05 гипотеза H_0 не отвергается

Для каких истинных значений p с точки зрения практики можно считать, что связь между черным котом и неудачей есть? Теперь сгенерируйте 10 выборок для двух случаев: 1). $n = 5, p = 0.75$; 2). $n = 10^5, p = 0.51$. В каждом случае в виде таблицы выведите реализацию статистики $T(X)$, соответствующее p-value и 0/1 - отвергается ли H_0 (выводите 1, если отвергается). Какие выводы можно сделать?

In [1116]:
```
n_array = [5, 10**5]
p_array = [0.75, 0.51]
alpha = 0.05
statistics = []
for n, p in zip(n_array, p_array):
    t = np.sum(sps.bernoulli(p=p).rvs(size=n))
    c = sps.binom(n=n, p=0.5).ppf(1 - alpha)
    pvalue = round(sps.binom(n, 0.5).sf(t), 3)
    statistics.append([t, pvalue, c])
stats = pd.DataFrame(data=statistics, columns=['t', 'p-value', 'c'])
stats['отвергаем $H_0$'] = list(map(lambda x: int(x), stats['p-value'] <= alpha))
stats
```

Out[1116]:

	t	p-value	c	отвергаем H_0
0	5	0.0	4.0	1
1	51099	0.0	50260.0	1

Тут мы столкнулись с влиянием размера выборки:

при малом размере выборки, мы получим, что функция вероятности нашей статистики имеет более сплюснутый вид, поэтому чтобы получить статистику с p-value, достаточным для отвержения H_0, нам нужно далеко уйти от среднего значения статистики

при большом размере выборки, наоборот, основная вероятностная масса сконцентрирована в окрестности 0.5, поэтому даже незначительное отклонение от 0.5 нашей статистики приводит к тому, что вероятность ее получения становится очнь маленькой.

Возникает задача подбора оптимального размера выборки.

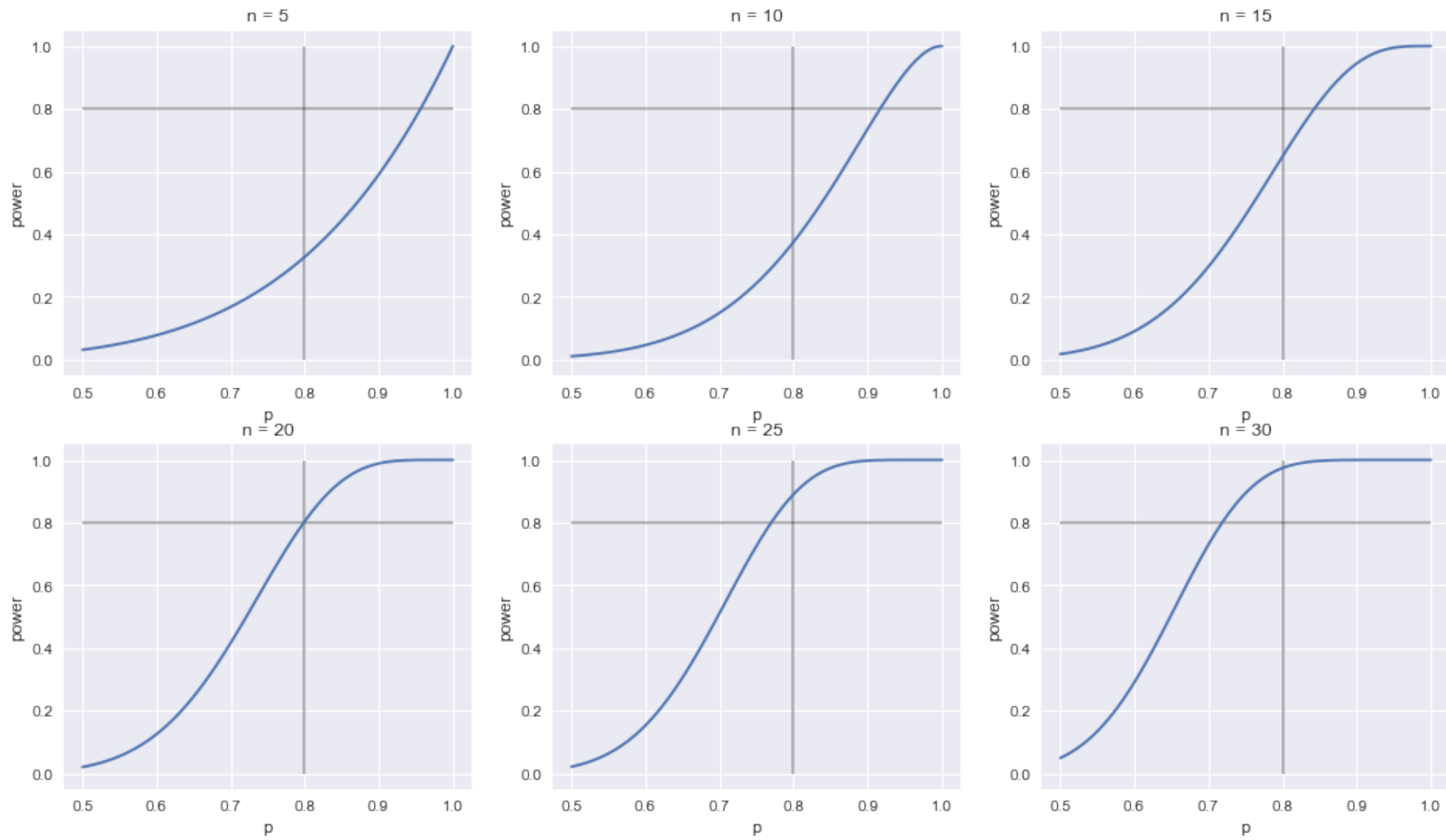
Для этого сначала зафиксируйте значение $p^* > 1/2$, которое будет обладать следующим свойством. Если истинное $p > p^*$, то такое отклонение от $1/2$ с практической точки зрения признается существенным, то есть действительно чаще случается неудача после того, как черный кот перебегает дорогу. В противном случае отклонение с практической точки зрения признается несущественным.

Теперь для некоторых n постройте графики функции мощности критерия при $1/2 < p < 1$ и уровне значимости 0.05. Выберите такое n^*, для которого функция мощности дает значение 0.8 при p^*. Для выбранного n^* проведите эксперимент, аналогичный проведенным ранее экспериментам, сгенерировав выборки для следующих истинных значений p: 1). $1/2 < p < p^*$; 2). $p > p^*$. Сделайте вывод.

Зафиксируем $p^* = 0.8$

In [1117]:
```
p_star = 0.8
beta = 0.8
alpha = 0.05
def power(n, p_star, alpha):
    return sps.binom.sf(sps.binom(n=n, p=0.5).ppf(1 - alpha), n=n, p=p_star)

n_array = [5, 10, 15, 20, 25, 30]
p = np.linspace(0.5, 1, 100)
plt.figure(figsize=(16, 9))
for i, n in zip(np.arange(1, len(n_array) + 1), n_array):
    plt.subplot(2, 3, i)
    plt.plot(p, power(n, p, alpha))
    plt.title('n = {}'.format(n))
    plt.hlines(beta, 0.5, 1, alpha=0.3)
    plt.vlines(p_star, 0, 1, alpha=0.3)
    plt.xlabel('p')
    plt.ylabel('power')
plt.show()
```



получили, что при $n^* = 20$ функция мощности дает значение 0.8 при p^*

рассмотрим подробнее, что же получается при n^* -- оптимальном размере выборки для проверки гипотезы H_0 vs H_1, подобранным с помощью эвристики на мощность критерия.

In [1133]:
```
n_star = 20
p_array = [0.6, 0.9]
alpha = 0.05
statistics = []
for p in p_array:
    t = np.sum(sps.bernoulli(p=p).rvs(size=n_star))
    c = sps.binom(n=n_star, p=0.5).ppf(1 - alpha)
    pvalue = round(sps.binom(n=n_star, p=0.5).sf(t), 4)
    statistics.append([t, pvalue, c])
stats = pd.DataFrame(data=statistics, columns=['t', 'p-value', 'c'])
stats['отвергаем $H_0$'] = list(map(lambda x: int(x), stats['p-value'] <= alpha))
stats
```

Out[1133]:

	t	p-value	c	отвергаем H_0
0	10	0.4119	14.0	0
1	17	0.0002	14.0	1

Вывод: С правильно подобранным размером выборки критерии не такие чувствительные, как при выборках с большим размером и не такие "пофигистичные", как в выборках малого размера

Справка для выполнения следующих задач

Критерий согласия хи-квадрат

`scipy.stats.chisquare` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chisquare.html#scipy.stats.chisquare)`(f_obs, f_exp=None, ddof=0)`

`f_obs` --- число элементов выборки, попавших в каждый из интервалов

`f_exp` --- ожидаемое число элементов выборки (по умолчанию равномерное)

`ddof` --- поправка на число степеней свободы. Статистика асимптотически будет иметь распределение хи-квадрат с числом степеней свободы $k - 1 - ddof$, где k --- число интервалов.

Возвращает значение статистики критерия и соответствующее p-value.

Критерий согласия Колмогорова

`scipy.stats.kstest` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kstest.html#scipy.stats.kstest)`(rvs, cdf, args=())`

`rvs` --- выборка

`cdf` --- функция распределения (сама функция или ее название)

`args` --- параметры распределения

Возвращает значение статистики критерия и соответствующее p-value.

Задача 7.

- Проверьте, что ваша выборка значений скорости ветра из задания 2 действительно согласуется с распределением Вейбулла.
- Проверьте, что при больших n распределение статистики из задач 3 и 4 задания 2 действительно хорошо приближают предельное распределение.
- Проверьте, что остатки в регрессии из задач выше нормальны.
- Подберите класс распределений для выборки количества друзей из задания 1.

Использовать можно два описанных выше критерия, либо любой другой критерий, если будет обоснована необходимость его применения в данной задаче, а так же будет приведено краткое описание критерия. Уровень значимости взять равным 0.05.

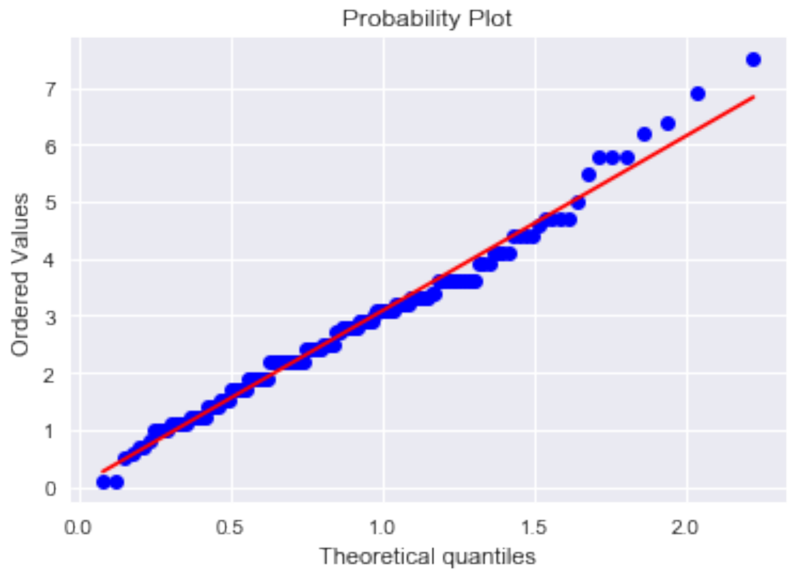
Мне очень понравился `QQ-plot`, поэтому в каждой из подзадач мы будем смотреть вначале на него и делать какие-то промежуточные выводы, а потом смотреть на более математичные статистики. Вообще я согласен с принципом "бикини": нужно чаще просто смотреть на графики/гистограммы распределений выборки, чтобы сделать какие-то выводы, а не бездумно использовать формальные статистические критерии

Проверьте, что ваша выборка значений скорости ветра из задания 2 действительно согласуется с распределением Вейбулла.

Взял парамметры значений скорости ветра и подобранные по сетке параметры распределения Вейбулла

In [1175]:
```
wind_speed = np.array([
    2.2, 3.3, 1.1, 0.8, 2.7, 2.8, 3.4, 3.6, 3.9, 2.2, 1.9, 2.4, 3.3, 2.2, 1.1,
    1.9, 0.6, 2.2, 1.9, 2.5, 1.7, 1.9, 1.1, 1.9, 3.1, 1.7, 1.0, 2.9, 2.8, 4.7,
    3.1, 2.4, 1.9, 2.8, 3.1, 2.8, 4.7, 3.1, 3.1, 4.6, 4.4, 1.2, 3.6, 5.8, 6.4,
    4.1, 2.2, 2.9, 1.7, 1.5, 1.5, 1.1, 3.1, 3.6, 3.9, 3.6, 3.4, 2.9, 3.2, 0.5,
    1.0, 2.5, 3.6, 5.5, 5.0, 3.9, 1.1, 2.2, 2.2, 4.1, 1.9, 3.6, 5.8, 6.2, 3.2,
    2.5, 1.4, 0.1, 2.2, 3.6, 4.7, 5.8, 4.4, 3.6, 4.1, 4.4, 3.2, 2.4, 1.7, 1.7,
    2.9, 3.2, 0.7, 1.0, 1.9, 3.1, 4.4, 3.3, 1.4, 0.1, 2.5, 2.9, 0.7, 2.2, 7.5,
    6.9, 4.7, 1.7, 4.1, 1.4, 1.5, 2.4, 1.7, 2.4, 2.8, 3.3, 2.7, 3.6, 2.2, 2.8,
    2.5, 2.4, 2.4, 3.3, 1.2, 3.0, 3.2, 3.6, 2.2, 2.2, 2.9, 3.6, 1.9, 1.2, 2.4,
    1.4, 1.5, 1.2, 2.2, 1.2, 3.3, 2.2, 2.8, 3.3, 1.0, 1.2, 2.7, 2.2
])
c =  2.10701; scale =  3.08161
```

In [1177]:
```
res = sps.probplot(x=wind_speed, dist=sps.weibull_min(c=c), plot=plt)
```



Ну очень хорошо же, судя по `QQ-plot`, посмотрим, что нам скажет Колмогоров:

In [1136]:
```
sps.kstest(wind_speed, sps.weibull_min(c=c, scale=scale).cdf)
```

Out[1136]:
```
KstestResult(statistic=0.080745955882075782, pvalue=0.27512181350825449)
```

Вывод: Действительно, распределение ветра согласуется с распределением Вейбулла с праматрами, подобранными по сетке

Проверьте, что при больших n распределение статистики из задач 3 и 4 задания 2 действительно хорошо приближают предельное распределение.

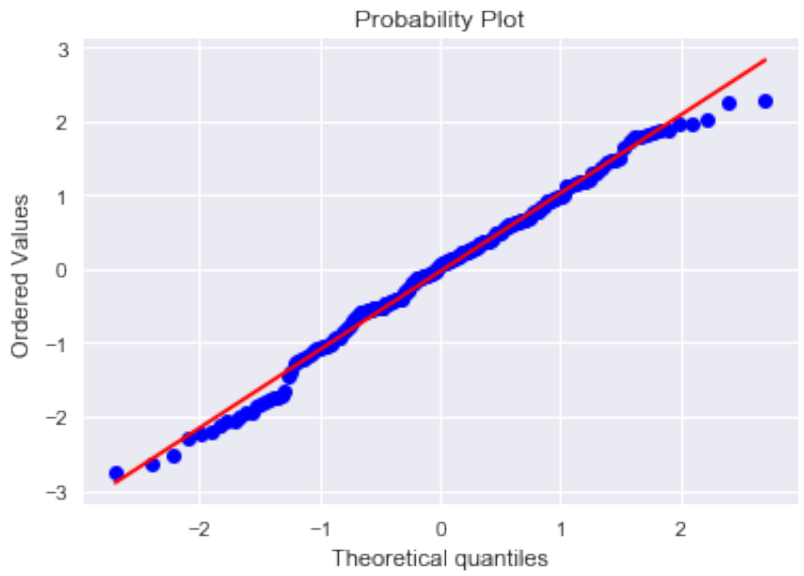
Задача 3 из задания 2

а) Сгенерируйте 200 выборок $X_1^j, \ldots, X_{300}^j$ из распределения $\mathcal{N}(0, 1)$. По каждой из них посчитайте оценки $\widehat{\theta}_{jn} = \frac{1}{n}\sum\limits_{i=1}^{n} X_i^j$ для $1 \leqslant n \leqslant 300$, то есть оценка параметра по первым n наблюдениям j-й выборки. Для этой оценки посчитайте статистику $T_{jn} = \sqrt{n}\left(\widehat{\theta}_{jn} - \theta\right)$, где $\theta = 0$.

In [1180]:
```
cnt_samples = 200
n = 300
theta = 0
samples = sps.norm.rvs(size=(cnt_samples, n))
theta_jn = np.cumsum(samples, axis=1) / np.arange(1, n + 1)
T_jn = np.sqrt(np.arange(1, n + 1)) * (theta_jn - theta)
T_sample = T_jn[:, -1]
sps.kstest(T_sample, sps.norm.cdf)
```

Out[1180]:
```
KstestResult(statistic=0.051749253533634444, pvalue=0.66232387164922191)
```

In [1181]:
```
res = sps.probplot(x=T_sample, dist=sps.norm, plot=plt)
```



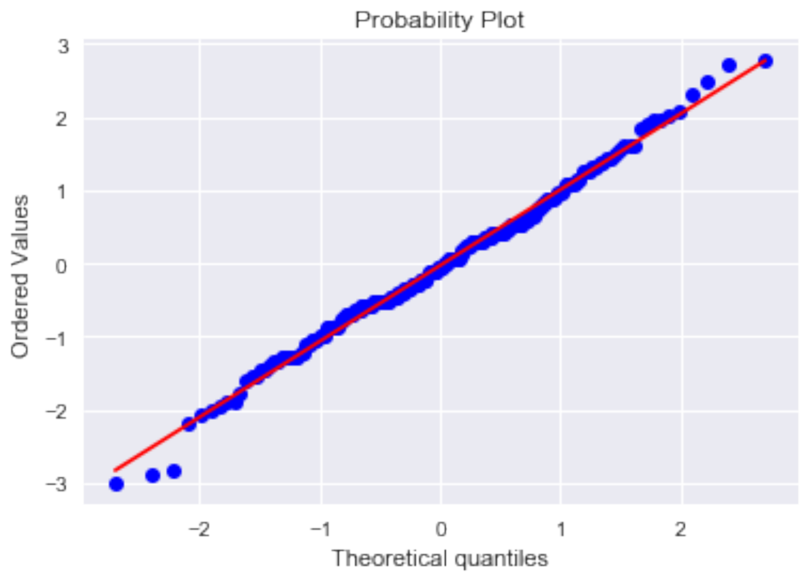
Вывод: Действительно, распределение из пункта а) согласуется с $\mathcal{N}(0, 1)$

b). Пусть $X_1, \dots, X_n$ --- выборка из распределения $Pois(\theta)$. Известно, что $\overline{X}$ является асимптотически нормальной оценкой параметра θ.

In [1182]:
```
cnt_samples = 200
n = 300
theta = 1
samples = sps.poisson(mu=theta).rvs(size=(cnt_samples, n))
theta_jn = np.cumsum(samples, axis=1) / np.arange(1, n + 1)
T_jn = np.sqrt(np.arange(1, n + 1)) * (theta_jn - theta)
T_sample = T_jn[:, -1]
sps.kstest(T_sample, sps.norm.cdf)
```

Out[1182]:
```
KstestResult(statistic=0.056665886145933153, pvalue=0.53336856312975556)
```

In [1183]:
```
res = sps.probplot(x=T_sample, dist=sps.norm, plot=plt)
```



Вывод: Действительно, распределение из пункта б) согласуется с $\mathcal{N}(0, 1)$

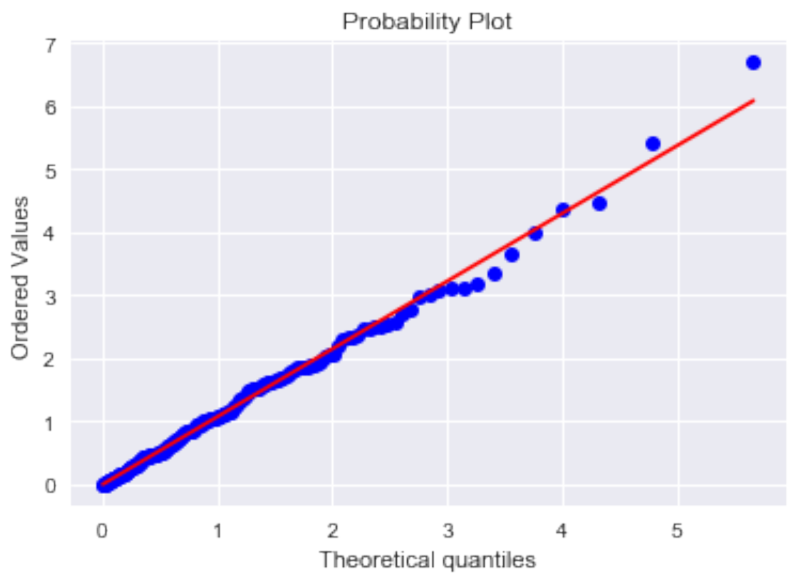
Задача 4 из задания 2

Пусть $X_1, \ldots, X_n$ --- выборка из распределения $U[0, \theta]$. Из домашнего задания известно, что $n\left(\theta - X_{(n)}\right) \xrightarrow{d_\theta} Exp\,(1/\theta)$.

In [1184]:
```
cnt_samples = 200
n = 300
theta = 1
samples = sps.uniform.rvs(size=(cnt_samples, n))
estim = (theta - np.maximum.accumulate(samples, axis=1)) * np.arange(1, n + 1)
T_sample = estim[:, -1]
sps.kstest(T_sample, sps.expon(scale=1).cdf)
```

Out[1184]:
```
KstestResult(statistic=0.05954210594183007, pvalue=0.46570020127316658)
```

In [1187]:
```
res = sps.probplot(x=T_sample, dist=sps.expon, plot=plt)
```



Вывод: Действительно, распределение из 4 задания согласуется с $Exp\,(1/\theta)$

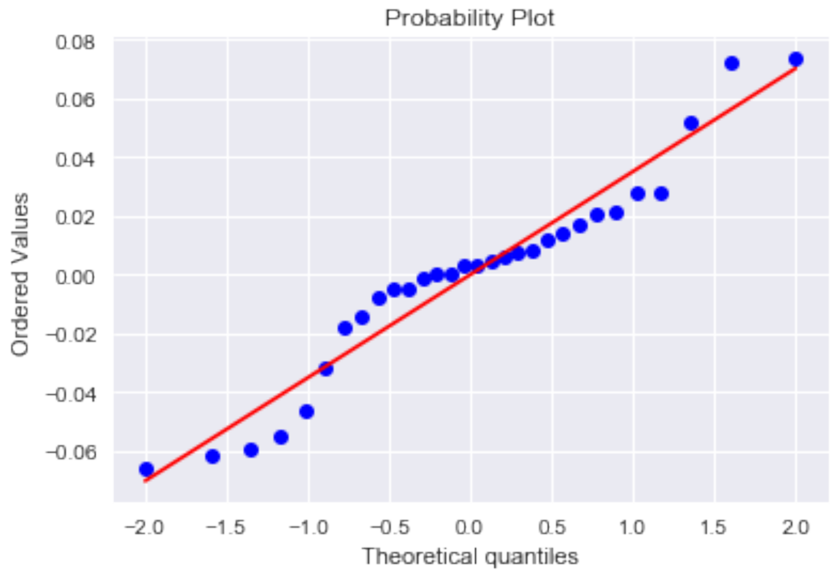
Проверьте, что остатки в регрессии из задач выше нормальны.

Возьмем остатки из задачи про мороженное

In [1149]:
```
df = pd.read_csv('ice_cream.txt', delimiter='\t')
df['temp'] = df['temp'].apply(lambda x: (x - 32) / 1.8)
X = df.drop(['date', 'Year', 'IC'], axis=1).values
y = df['IC'].values
est = LinearRegression()
est.fit(X, y)
remainders = y - est.predict(X)
```

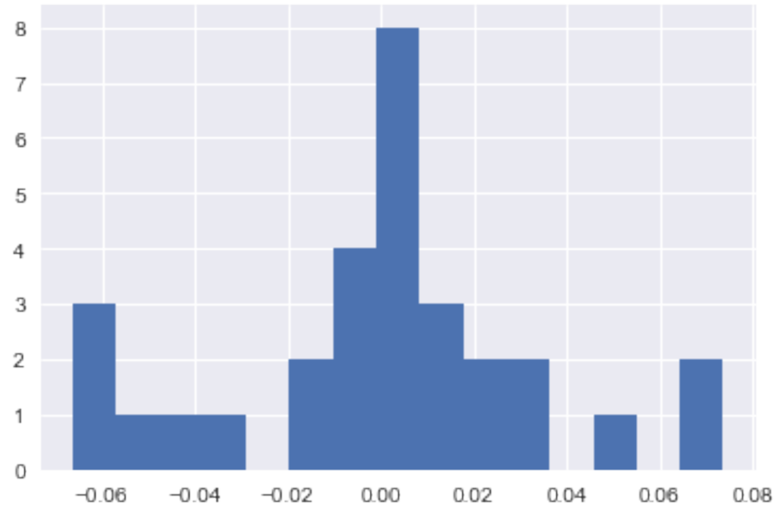
Посмотрим вначале на `QQ-plot`:

In [1150]:
```
res = sps.probplot(x=remainders, dist=sps.norm, plot=plt)
```



Посмотрим на гистограмму остатков:

In [1188]:
```
plt.hist(remainders, bins=15)
plt.show()
```



Ну какие-то зачатки нормального распределения все-таки есть

Теперь проверим гипотезу H_0: остатки распределены нормально, для этого воспользуемся критерием Шапиро-Уилка, описанным ниже

In [1189]:
```
W, p_value = sps.shapiro(remainders)
print('W = {}, p-value = {}'.format(W, p_value))
```

```
W = 0.9414945244789124, p-value = 0.09979545325040817
```

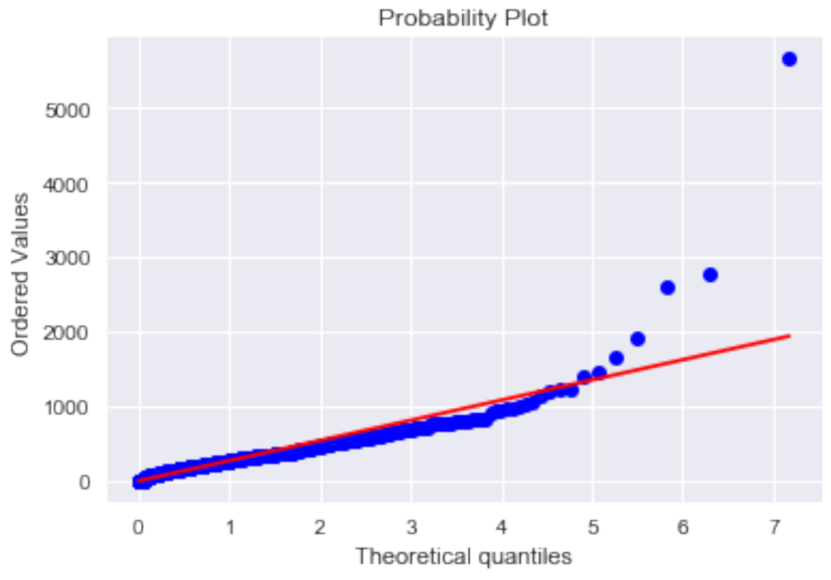
Вывод: Действительно, значение p-value получилось достаточное, поэтому мы не можем отвергнуть гипотезу H_0 и я все-таки бы сказал, что остатки регрессии распределены нормально

Подберите класс распределений для выборки количества друзей из задания 1.

In [846]:
```
sample_vk = [344, 1229, 1141, 1037, 535, 1238, 372, 194, 166, 299, 280, 471, 709, 715, 2764,
             163, 97, 779, 498, 606, 156, 418, 224, 730, 137, 1191, 525, 387, 763, 358, 583,
             250, 51, 148, 392, 1464, 540, 253, 621, 291, 1064, 123, 284, 264, 916, 546, 308,
             335, 1911, 493, 230, 503, 463, 265, 150, 713, 938, 580, 451, 218, 111, 187, 226,
             103, 218, 321, 266, 70, 673, 314, 482, 516, 547, 316, 604, 133, 231, 148, 716,
             548, 771, 320, 323, 148, 274, 141, 197, 270, 346, 87, 147, 180, 129, 565, 776,
             507, 338, 118, 135, 273, 321, 211, 257, 132, 657, 136, 453, 288, 200, 513, 279,
             258, 522, 301, 338, 342, 636, 178, 440, 538, 249, 200, 111, 92, 86, 570, 321,
             154, 532, 135, 144, 107, 160, 103, 213, 164, 257, 225, 131, 97, 279, 651, 304,
             161, 2, 71, 139, 222, 564, 180, 118, 220, 402, 254, 522, 437, 653, 121, 206, 171,
             421, 1391, 764, 73, 143, 56, 423, 229, 149, 193, 350, 478, 418, 261, 126, 136, 297,
             264, 677, 584, 225, 323, 343, 250, 295, 217, 176, 205, 204, 332, 122, 317, 5660,
             506, 488, 471, 277, 124, 295, 61, 88, 70, 400, 194, 132, 195, 79, 444, 279, 429,
             681, 445, 553, 163, 128, 86, 529, 247, 176, 248, 190, 343, 20, 339, 331, 61, 348,
             292, 353, 159, 128, 241, 370, 242, 270, 502, 699, 808, 713, 196, 427, 330, 547,
             425, 315, 152, 80, 352, 217, 247, 272, 214, 431, 217, 272, 135, 362, 183, 111, 253,
             364, 152, 532, 180, 154, 60, 191, 436, 332, 99, 79, 370, 353, 558, 495, 481, 581,
             363, 316, 202, 499, 825, 229, 265, 263, 813, 481, 46, 722, 327, 94, 330, 196, 305,
             260, 150, 264, 145, 206, 371, 480, 406, 83, 448, 260, 216, 422, 456, 344, 365, 309,
             175, 806, 331, 339, 293, 157, 183, 59, 307, 455, 118, 215, 204, 76, 119, 156, 217,
             366, 359, 201, 275, 417, 404, 1010, 102, 308, 109, 271, 466, 208, 326, 263, 369, 398,
             157, 124, 86, 330, 56, 529, 565, 347, 148, 0, 553, 268, 261, 329, 612, 319, 87, 110,
             272, 974, 231, 178, 109, 353, 312, 660, 145, 31, 206, 792, 122, 387, 172, 127, 71,
             114, 92, 90, 169, 135, 482, 132, 363, 224, 166, 58, 83, 340, 358, 178, 166, 586, 38,
             97, 277, 0, 95, 263, 116, 193, 6, 89, 258, 147, 112, 50, 208, 230, 137, 353, 377, 439,
             87, 180, 355, 150, 67, 139, 252, 203, 184, 372, 305, 119, 324, 256, 214, 182, 109, 269,
             240, 167, 189, 215, 190, 103, 811, 617, 78, 263, 167, 222, 115, 183, 96, 198, 326, 354,
             178, 141, 109, 204, 120, 525, 658, 206, 44, 82, 24, 563, 563, 789, 385, 297, 4, 180,
             75, 107, 0, 233, 178, 119, 66, 43, 258, 377, 242, 932, 158, 96, 675, 326, 32, 447, 57,
             209, 120, 233, 502, 200, 147, 243, 70, 339, 331, 67, 198, 602, 193, 299, 375, 118, 138,
             511, 86, 95, 315, 137, 661, 159, 272, 310, 54, 351, 309, 43, 350, 436, 105, 682, 304, 0,
             348, 50, 143, 155, 180, 813, 224, 336, 369, 273, 73, 131, 249, 260, 476, 89, 347, 135,
             217, 377, 161, 202, 144, 230, 325, 319, 171, 187, 375, 94, 131, 118, 256, 89, 241, 346,
             143, 271, 359, 185, 102, 768, 206, 339, 131, 194, 359, 244, 117, 159, 136, 283, 504,
             207, 275, 249, 173, 113, 183, 243, 226, 313, 134, 300, 81, 183, 77, 213, 160, 135, 36,
             195, 199, 136, 291, 176, 0, 186, 223, 144, 139, 395, 316, 271, 330, 668, 303, 114, 254,
             131, 423, 195, 32, 76, 122, 141, 630, 1, 89, 216, 776, 34, 164, 169, 131, 352, 209, 109,
             134, 0, 250, 348, 245, 52, 229, 222, 209, 552, 112, 206, 136, 62, 320, 212, 153, 175,
             83, 112, 203, 194, 166, 473, 83, 47, 243, 260, 794, 163, 91, 380, 357, 1653, 67, 128,
             188, 429, 69, 178, 81, 584, 169, 353, 68, 159, 87, 51, 113, 130, 185, 506, 244, 313, 0,
             126, 68, 289, 290, 213, 49, 166, 199, 25, 371, 253, 281, 262, 215, 188, 333, 54, 283,
             189, 436, 311, 72, 187, 269, 219, 221, 771, 390, 402, 104, 135, 141, 211, 271, 429,
             300, 165, 91, 313, 167, 149, 450, 152, 129, 90, 187, 222, 53, 241, 78, 171, 135, 169,
             225, 71, 78, 164, 15, 0, 38, 0, 184, 323, 28, 147, 24, 0, 63, 266, 244, 64, 130, 7, 59,
             52, 376, 313, 298, 172, 267, 59, 120, 12, 134, 170, 288, 319, 194, 86, 156, 327, 73, 52,
             154, 16, 144, 137, 204, 166, 284, 77, 145, 313, 67, 252, 42, 162, 134, 220, 0, 420, 75,
             132, 163, 205, 15, 70, 39, 60, 964, 447, 234, 421, 104, 77, 98, 13, 0, 0, 297, 109, 116,
             28, 84, 50, 69, 154, 20, 77, 153, 1, 82, 42, 209, 277, 147, 138, 1, 1, 0, 174, 312, 21,
             158, 9, 157, 221, 2, 5, 615, 2, 71, 216, 152, 0, 221, 1, 101, 52, 7, 0, 125, 10, 2605, 71,
             80, 42, 0, 62, 0, 96, 206, 84, 1, 76, 98, 0, 74, 4, 30, 185, 7, 116, 230, 71, 97, 174, 32,
             114, 1, 18, 92, 73, 82, 74, 9, 0, 28, 0, 31, 0, 6, 0, 1, 2, 29]
```

Давайте вначале посмотрим на `QQ-plot`

In [857]:
```
res = sps.probplot(x=sample_vk, dist=sps.expon, plot=plt)
```



Что-то похожее на экспоненциальное (я это исследовал в 1ом задании), но приближение не очень точное, давайте посмотрим на какой-нибудь более точный критерий:

In [875]:
```
sps.kstest(sample_vk, sps.expon(scale=1 / 0.00389738973897).cdf)
```

Out[875]:
```
KstestResult(statistic=0.11652899719967086, pvalue=3.5597969016976094e-11)
```

Критерий говорит, что это не экспоненциальное распределение, но я бы сослался все-таки на некоторую специфичность группы `inter oves locum praesta`: можно сказать, что у них достаточно мало интровертов (это видно из последних квантилей `QQ-plot`) и из-за этого гипотеза о том, что распределение друзей -- экспоненцильное распределение отвергается.

Задача 8*. Проведите исследование согласно примеру 2 параграфа 2 главы 18 книги М.Б. Лагутина "Наглядная математическая статистика".

In [1156]:
```
M = 537 # общее число падений
n = 24 * 24 # общее число клеток, на которые разделен Лондон
theta = M / n
l = [229, 211, 93, 35, 7, 0, 0, 1]  # число падений (на i-ом месте количество учасков,
                                    # на которые упало i+1 снарядов)
```

In [1157]:
```
expectation_values = [ n * sps.poisson(mu=theta).pmf(i) for i in range(len(l))]
expectation_values
```

Out[1157]:
```
[226.74272258323953,
 211.39035074166597,
 98.538731205099509,
 30.622279315473637,
 7.1372239550387757,
 1.3307948832832714,
 0.20678149662127213,
 0.027540095160124806]
```

In [1158]:
```
upd_expectation_values = expectation_values[:4] + [sum(expectation_values[4:])]
print(upd_expectation_values)
```

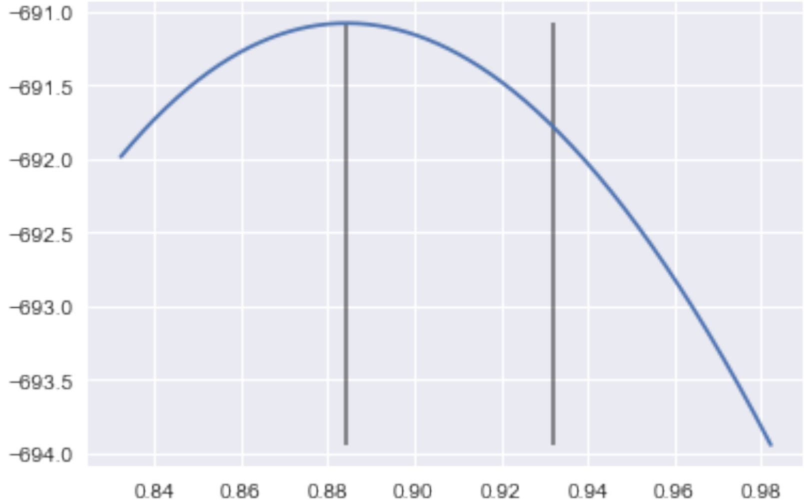
```
[226.74272258323953, 211.39035074166597, 98.538731205099509, 30.622279315473637, 8.7023404301034439]
```

Помним, что для использования критерия хи-квадрат нам нужно иметь значения $n\pi_j(\widehat{\theta}) > 5$. Для этого объединим последние 4 слолбца и будем максимизировать по θ функцию

$$\sum_{j=1}^{N} l_j \ln p_j(\theta)$$

In [1159]:
```
def likelihood(theta):
    p_from_1_to_4 = sps.poisson(mu=theta).pmf(np.arange(4))
    p_5 = 1 - np.sum(p_from_1_to_4)
    response = np.sum(np.array(l[:4]) * np.log(p_from_1_to_4)) + np.sum(l[4:]) * p_5
    return response
```

In [1160]:
```
grid_theta = np.linspace(theta - 0.1, theta + 0.05, 1000)
likelihood_values = np.array(list(map(lambda x: likelihood(x), grid_theta)))
best_theta = grid_theta[np.argmax(likelihood_values)]
plt.vlines(theta, min(likelihood_values), max(likelihood_values), alpha=0.5)
plt.vlines(best_theta, min(likelihood_values), max(likelihood_values), alpha=0.5)
plt.plot(grid_theta, likelihood_values)
plt.show()
```



Теперь пересчитаем ожидаемые значения количества клеток, в которые попало i снарядов

In [1161]:
```
best_theta_expectation_values = [ n * sps.poisson(mu=theta).pmf(i) for i in range(len(l))]
best_theta_expectation_values = best_theta_expectation_values[:4] + [sum(best_theta_expectation_values[4:])]
best_theta_expectation_values
```

Out[1161]:
```
[226.74272258323953,
 211.39035074166597,
 98.538731205099509,
 30.622279315473637,
 8.7023404301034439]
```

In [1162]:
```
experiment_data = np.array(l[:4] + [sum(l[4:])])
```

Теперь у нас есть все для того, чтобы по этим данным посчитать значение статистики хи-квадрат:

In [1163]:
```
sps.chisquare(f_obs=experiment_data, f_exp=best_theta_expectation_values, ddof=3)
```

Out[1163]:
```
Power_divergenceResult(statistic=1.0170342909820549, pvalue=0.31322351803519538)
```

In [1164]:
```
print("Наша статистика попадает в интервал, образованный 0.15 и 0.3 квантилями",
      "распределения хиквадрат: ({}, {})".format(round(sps.chi2(df=3).ppf(0.15), 3),
                                                 round(sps.chi2(df=3).ppf(0.30), 3)))
```

```
Наша статистика попадает в интервал, образованный 0.15 и 0.3 квантилями распределения хиквадрат: (0.798, 1
.424)
```

In [1165]:
```
sps.chi2(df=3).ppf(0.203)
```

Out[1165]:
```
1.0175726438594339
```

Если более точно, то фактический уровень значимости примерно 80%, поэтому мы можем принять гипотезу о низком уровне стрельбы

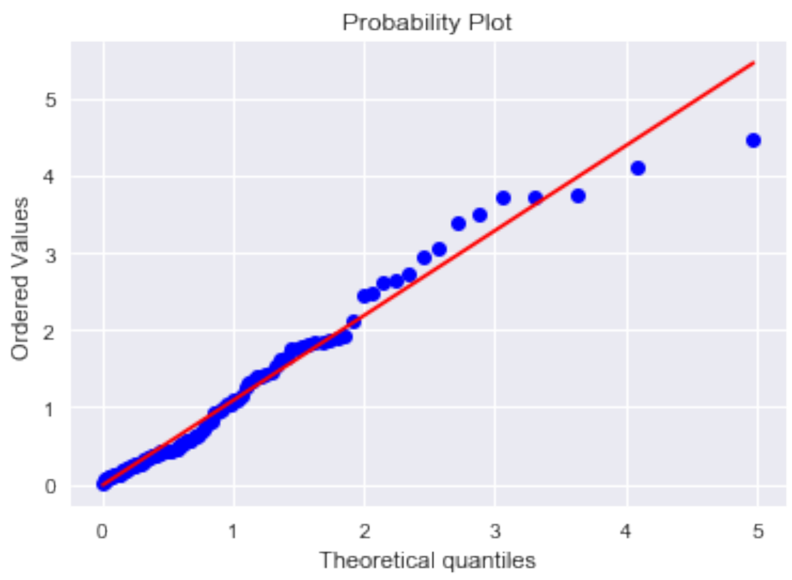
Задача 9*. Изучите Q-Q plot и критерий Шапиро-Уилка для проверки нормальности, напишите их теоретическое пояснение. В изучении могут помочь материалы курса ПСАД (http://wiki.cs.hse.ru/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B9_%D1%81%D1%82%

Постройте графики Q-Q plot для различных распределений и дайте к ним пояснение. Проверьте различные данные на нормальность с помощью различных критериев и Q-Q plot. Данные можно использовать из задачи 7 или какие-либо еще, например, отдельные компоненты из Ирисов Фишера. Постарайтесь так же правильно контролировать вероятность общей ошибки первого рода.

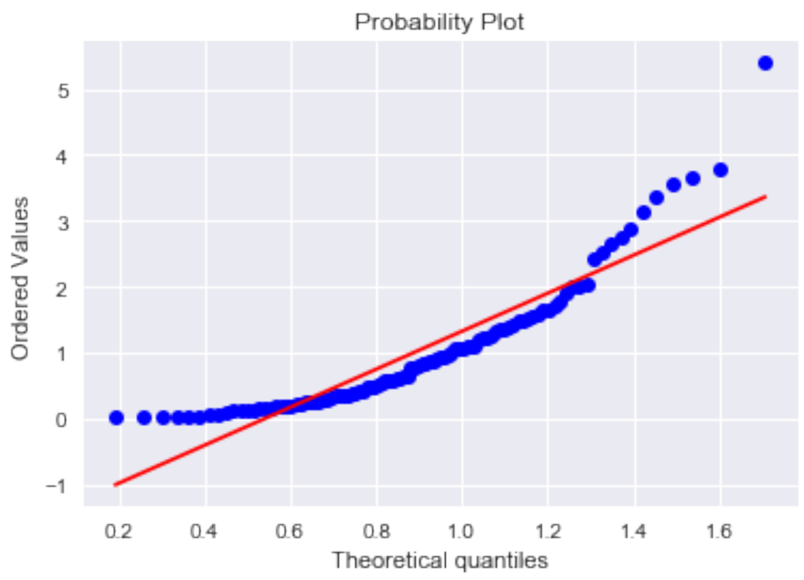
Q-Q plot -- график квантиль-квантиль. Этот график нужен для того, чтобы сравнить распределение случайной величины с каким-то теоретически известным. Он реализован в питоне функцией `scipy.stats.probplot`

Например, для выборки из экспоненциального, по `QQ-plot` мы действительно можем сказать (предположить, а далее использовать более математичные критерии проверки), что перед нами выборка из экспоненциального распределения

In [1166]:
```
sample = sps.expon.rvs(size=100)
res = sps.probplot(x=sample, dist=sps.expon, plot=plt)
```

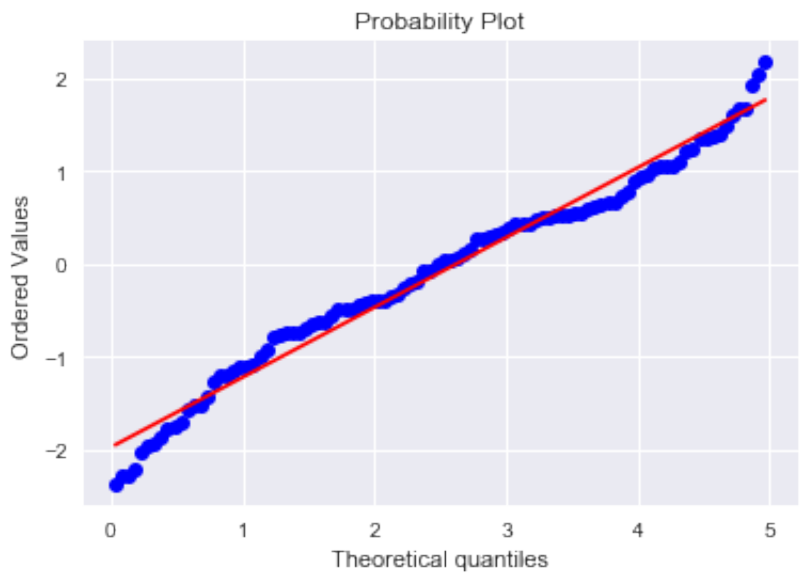


In [1167]:
```
sample = sps.expon.rvs(size=100)
res = sps.probplot(x=sample, dist=sps.weibull_min(c=3), plot=plt)
```



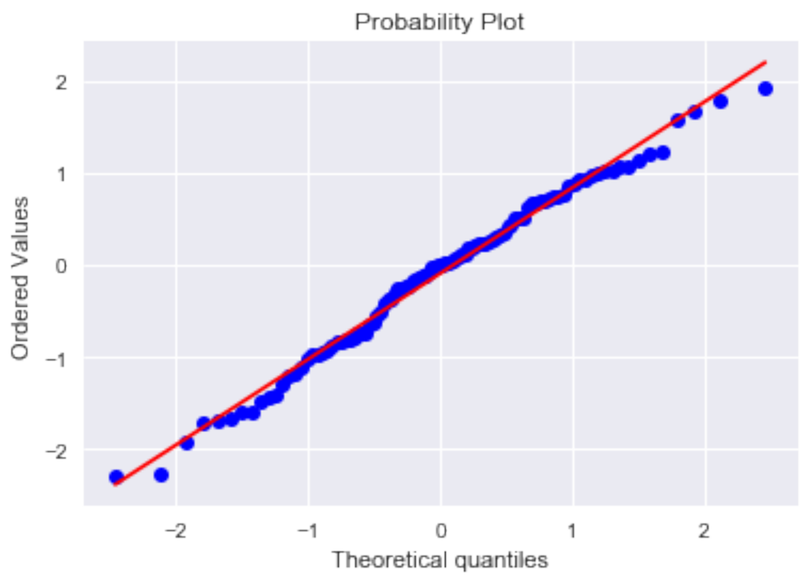
Действительно, получается достаточно наглядно, что выборка из стандартного нормального (чего мы не знаем) не является выборкой из равномерного на $[0, 5]$

In [1169]:
```
sample = sps.norm.rvs(size=100)
res = sps.probplot(x=sample, dist=sps.uniform(scale=5), plot=plt)
```



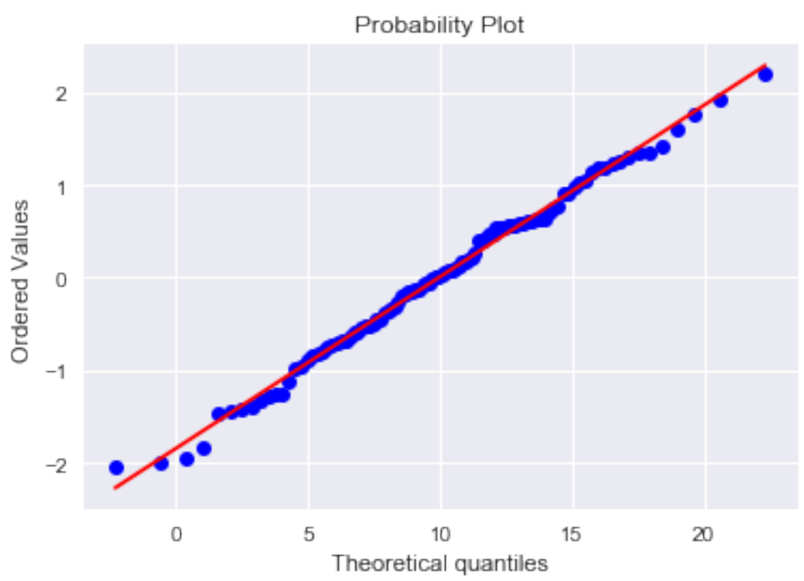
А вот, что это действительно является нормальным достаточно наглядно видно:)

In [1170]:
```
sample = sps.norm.rvs(size=100)
res = sps.probplot(x=sample, dist=sps.norm, plot=plt)
```



Но параметры подобранного распределения по `QQ-plot` мы не сможем оценить:

In [1171]:
```
sample = sps.norm.rvs(size=100)
res = sps.probplot(x=sample, dist=sps.norm(loc=10, scale=5), plot=plt)
```



Критерий Шапиро-Уилка используется для проверки гипотезы H_0 выборка X из нормального распределения. Этот критрий очень эффективен на проверку "нормальности" выборки

Статистика критерия есть:

$$W(X_1, \ldots, X_n) = \frac{\left(\sum_{i=1}^{n} c_{n-i+1}(x_{n-i+1} - x_i)\right)^2}{s^2},$$

где коэффициенты

$$c_{n-i+1} = \frac{m_{n-i+1}}{\left(\sum_{i=1}^{n} m_{i,n}^2\right)^{\frac{1}{2}}},$$

а $m_{i,n}$ -- математическое ожидание i-ой порядоквой стататистики стандартного нормального распределения.

Критическое значение $W(\alpha)$ также табулировано.

Если $W < W(\alpha)$, то нулевая гипотеза о нормальности распределения отклоняется при уровне значимости α.

Критерий Шапиро-Уилка реализован в `python` функцией `scipy.stats.shapiro`. Из документации в `scipy`:

The algorithm used is described in [R634] but censoring parameters as described are not implemented. For N > 5000 the W test statistic is accurate but the p-value may not be.

The chance of rejecting the null hypothesis when it is true is close to 5% regardless of sample size.

Проведем исследование с примером из задачи 7:

In [1172]:
```
cnt_samples = 200
n = 300
theta = 1
samples = sps.poisson(mu=theta).rvs(size=(cnt_samples, n))
theta_jn = np.cumsum(samples, axis=1) / np.arange(1, n + 1)
T_jn = np.sqrt(np.arange(1, n + 1)) * (theta_jn - theta)
T_sample = T_jn[:, -1]
W, p_value = sps.shapiro(T_sample)
print('W = {}, p-value = {}'.format(W, p_value))
```

```
W = 0.9926114678382874, p-value = 0.4113670289516449
```

На достаточно разумных уровнях значимости мы не можем отклонить гипотезу H_0 о нормальности распределения

Действительно, этот критерий очень крут на проверку нормальности выборки. Я протестировал его уже в предыдущих задачах и понял его крутость в сравнении с критерием Колмогорова. Но тут как и везде: не стоит всегда заглядываться на бикини, нужно посмотреть, что за ним (графики распределений)

In [ ]: