Using ABC method and MCMC to analyzing Ship speed data

1. Data description:

We use the data of the ship track and focus on the relation between speed and position of this ship. It has about 127078 rows and 15 features, each feature represent one column in the whole dataset. Cause we only focus on the positions effect to speed in this project, we only need to analyze the features 'LON' and 'LAT', which denote the longitude and latitude, respectively. First, let us upload csv file to python dataframe and print the first ten rows:

In [1]:
```
import pandas as pd
def read_data(file_name='ship.csv'):
    data = pd.read_csv('./code/{}'.format(file_name))
    data.dropna(how='any')
    return data

data = read_data()
print('The length of ship.csv data is {}'.format(len(data)))
data.head(10)
```

```
The length of ship.csv data is 127078
```

Out[1]:

	Var1	MMSI	SHIP_ID	STATUS	SPEED_KNOTSx10	LON	LAT	COURSE	HEADIN
0	727	276829000	4691590	5	0	24.770250	59.444149	15	206
1	736	276829000	4691590	5	0	24.770000	59.443329	203	Na
2	1628	276829000	4691590	0	226	24.696449	59.561600	159	160
3	1640	276829000	4691590	0	225	24.715771	59.663120	210	210
4	1641	276829000	4691590	0	224	24.744619	59.688480	209	210
5	1651	276829000	4691590	0	227	24.794331	59.772049	193	194
6	2601	276829000	4691590	0	225	24.716370	59.535412	158	160
7	2617	276829000	4691590	0	224	24.729851	59.675549	210	210
8	2623	276829000	4691590	0	227	24.773279	59.728069	193	194
9	3251	276829000	4691590	0	227	24.786650	59.756149	194	194

Then, we remove the unralted features and just concentrate on the SPEED, LON, LAT.

In [2]:
```
data = data[['SPEED_KNOTSx10', 'LON', 'LAT']]
data = data.rename(columns={'SPEED_KNOTSx10': 'SPEED'})
data.head(5)
```

Out[2]:

	SPEED	LON	LAT
0	0	24.770250	59.444149
1	0	24.770000	59.443329
2	226	24.696449	59.561600
3	225	24.715771	59.663120
4	224	24.744619	59.688480

Furthermore, we need to delete the rows that SPEED is equal to zero. Because when SPEED is zero that means ship are in on rest, we do not to regrad it as the information of sailing. Also, we need to divide the SPEED by 10 cause the current value of SPEED is the 10 times of speed knots.

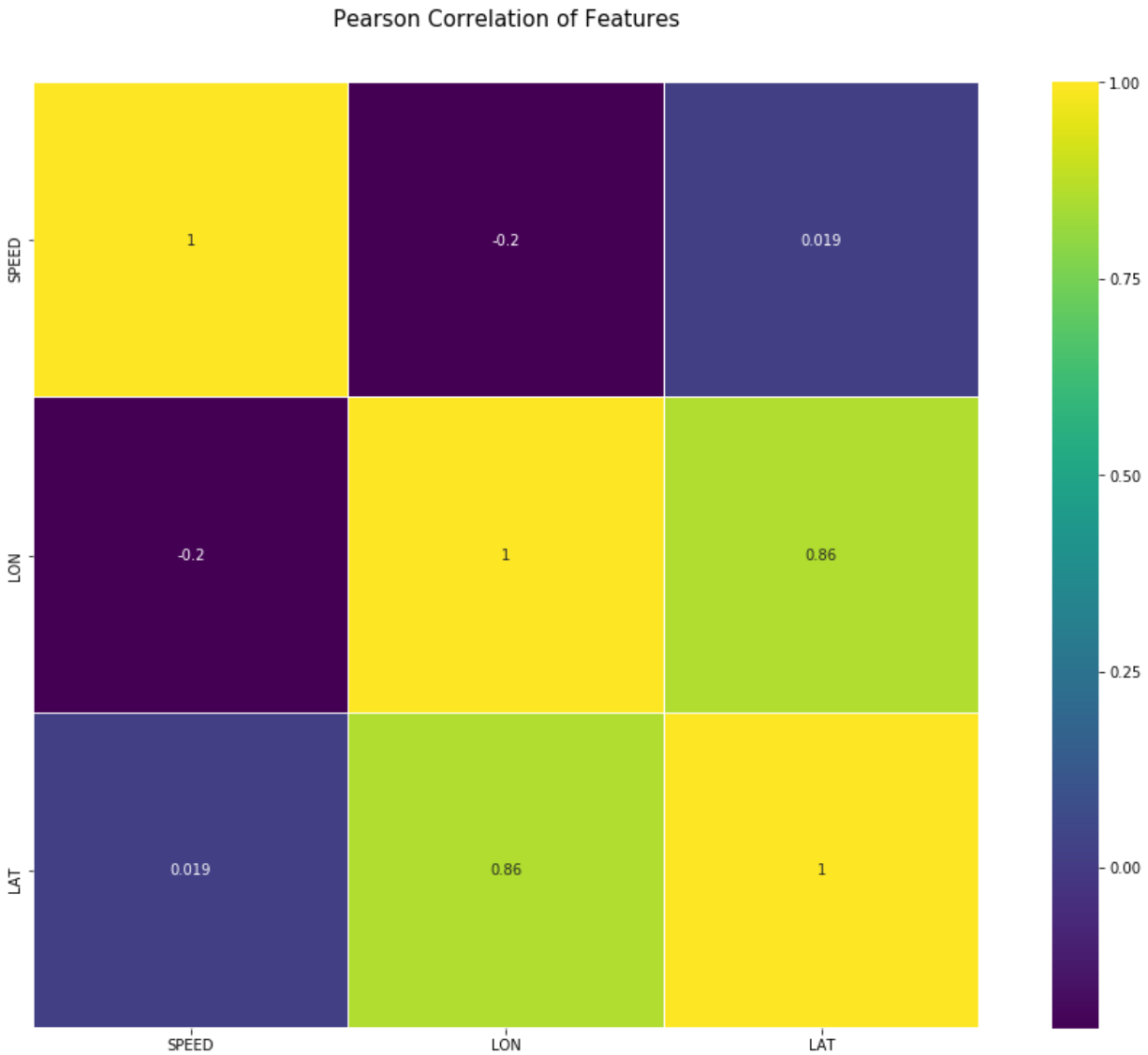
In [3]:
```
data = data[data.SPEED != 0]
data.SPEED /= 10

data.head(5)
```

Out[3]:

	SPEED	LON	LAT
2	22.6	24.696449	59.561600
3	22.5	24.715771	59.663120
4	22.4	24.744619	59.688480
5	22.7	24.794331	59.772049
6	22.5	24.716370	59.535412

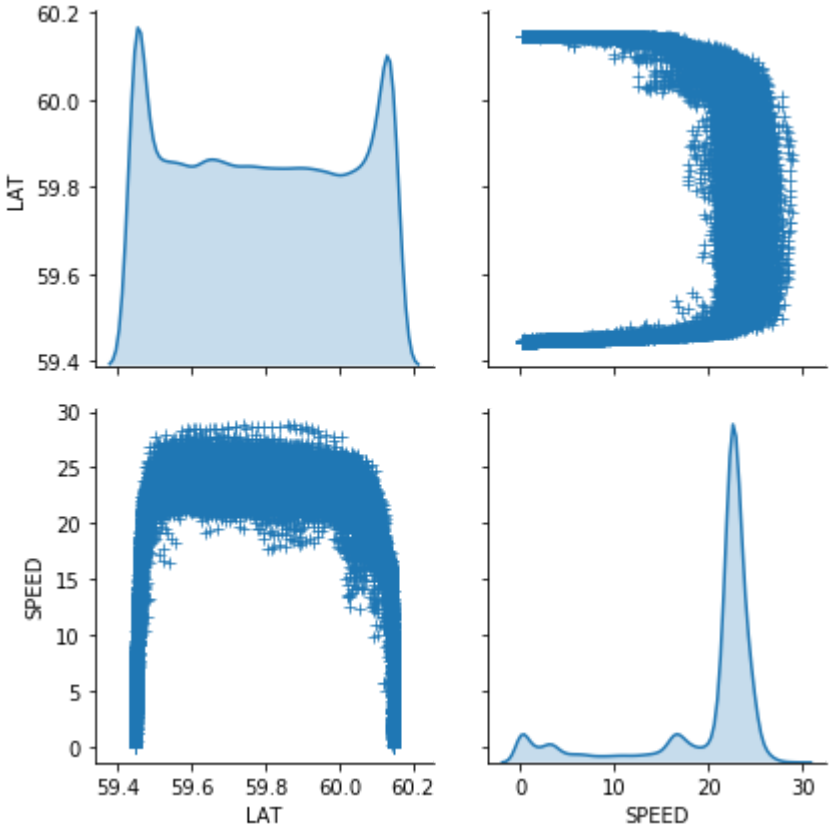
Now, we need to check if the LON and LAT have a strong relation with SPEED or not. In this part, we compute the **Pearson correlation** of these three columns and plot it:

In [4]:
```
import matplotlib.pyplot as plt
import seaborn as sns

def correlation(features):
    '''
    :param features: pd.dataframe
    :return: correlation matrix
    '''
    cor_matrix = features.corr(method="pearson")
    colormap = plt.cm.viridis
    plt.figure(figsize=(18, 12))
    plt.title('Pearson Correlation of Features', y=1.05, size=15)
    sns.heatmap(cor_matrix, linewidths=0.1, square=True, cmap=colormap, li
necolor='white', annot=True)
    plt.show()
    return cor_matrix

cor_matrix = correlation(data)
```



From the figure above we could know that LAT and LON are very relevant to each other with a high correlation value 0.86, which meets our daily experiences. As for the correlation values of SPEED and LON, LAT, although they are not very high, can still be analyzed because they actually have a slight relationship. We can use a pair plot to show the correlation of two features more clearly:

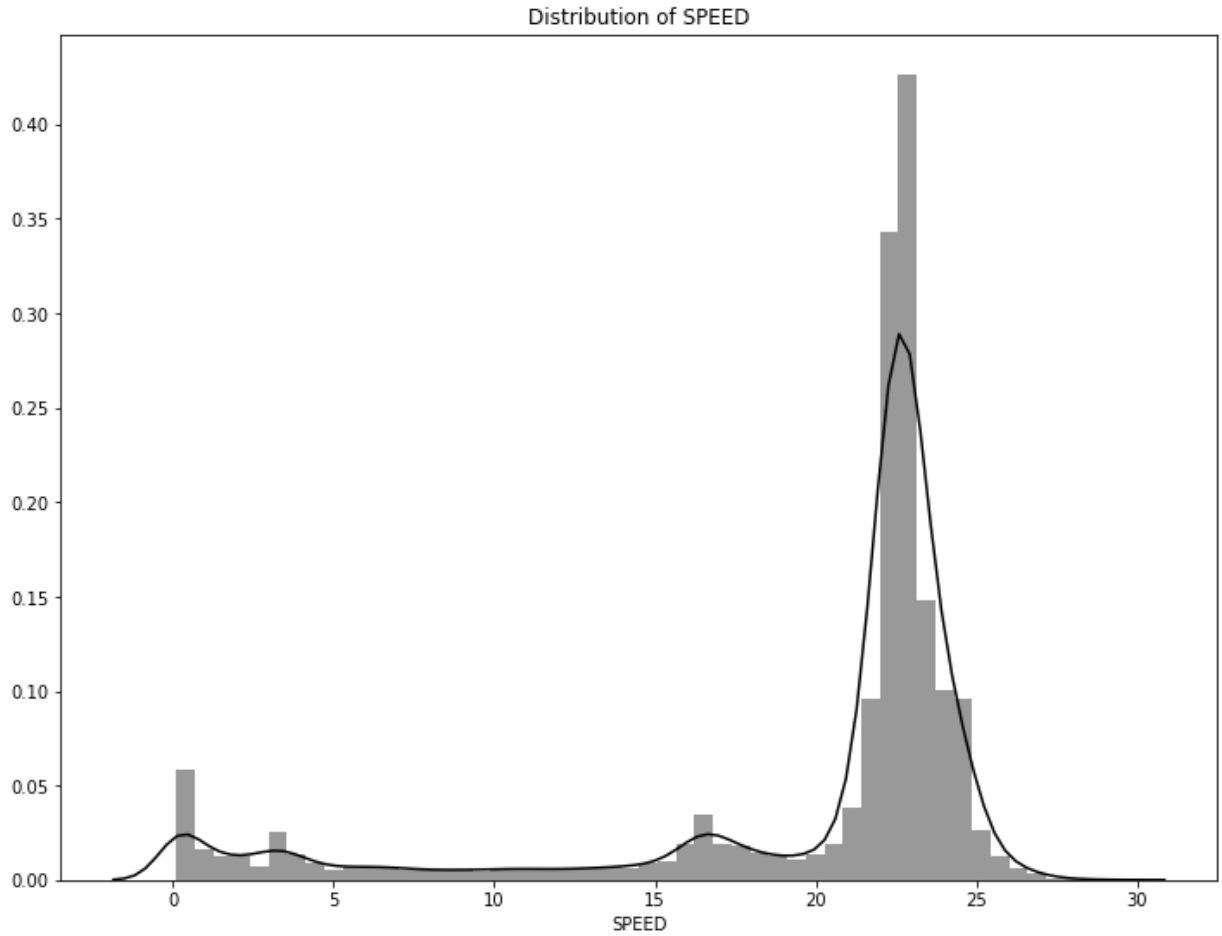
In [5]:
```
def plot_pair(data, pair):
    sns.pairplot(data, height=3, vars=pair, markers="+", diag_kind="kde")
    plt.show()

plot_pair(data, ('LAT', 'SPEED'))
```



From the scatter figure of LAT and SPEED, we can see a crude shape of normal distribution. It's really similar to the picture of normal distribution, so we can assume that our model of speed will probably a normal distribution.

Then, let us plot the prior distribution of SPEED, and check if it meets our assumptions:

In [6]:
```
def plot_prior(x):
    plt.figure(figsize=(12, 9))
    sns.distplot(x, bins=50, color='k', label=x.name)
    plt.title('Distribution of {}'.format(x.name))
    plt.show()

plot_prior(data['SPEED'])
```



From the plot of SPEED, we could detect that it's plot follow the shape of normal distribution in the intervals of [0,30] and [200, 280]. Furthermore, in the middle of the figure, it seems to follow a uniform distribution. Anyway, as this is only a project to test our knowledge in approximate bayesian Computation, so we don't need to pay lots time and attention to find a best model for our data. In this case, normal distribution for SPEED is neither a bad choice nor a good one, but let's us continue to do the task to see if our simulation of SPEED's posteprior will make a success.

2. Principal Components Analysis (PCA)

Principal Components Analysis (PCA) is a linear algebra technique that finds new variables (**principal components**) that have the following properties:

1. they are linear combinations of the original attributes
2. they are orthogonal (perpendicular) to each other
3. they describe all the variance of the original data.

The new *attributes* or *variables* derived from PCA better capture the variability of the data. More specifically, the first dimension retains as much of the variability as possible. The second dimension is orthogonal to the first, and, subject to that constraint, captures as much of the remaining variability as possible, and so on.

Since LAT and LON are actually have same variability and effect to the feature SPEED, we can use PCA method to combined the two alike features into one component without lose their variability. Also, in this case we can reduce the dimensionality of our data and speed up our analysing later.

2.1 Standardization and Normalization

Since our data attributes have different units, the difference between ones might be dominated by the difference between others: carbohydrates will range from 0 to 100, whereas energy values will range from few hundreds to 2 thousands. To take this into account, we first have to **standardize** our data - transform them so that each attribute has a **mean** of 0 and **standard deviation** 1. We standardize our data with the help of Python's Scikit-learn package's StandardScaler class.

In [7]:
```
from sklearn.preprocessing import StandardScaler
std_data = StandardScaler().fit_transform(data[['LON', 'LAT']].values)
std_data[:10]
```

Out[7]:
```
array([[-1.41697296, -1.0010633 ],
       [-1.16996984, -0.55786086],
       [-0.80119094, -0.44714756],
       [-0.16569673, -0.08231319],
       [-1.16231251, -1.11539137],
       [-0.98997792, -0.50359999],
       [-0.43481534, -0.27431519],
       [-0.26388693, -0.15172728],
       [ 0.10236084,  0.10690789],
       [-0.45373494, -1.50843668]])
```

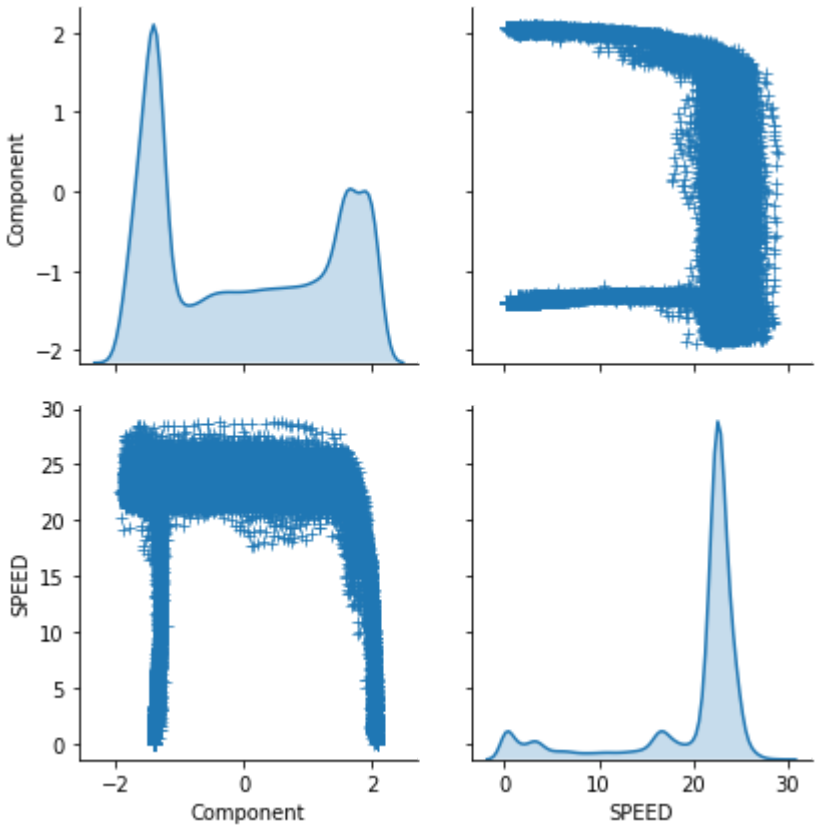
Now, let us still get the power of Scikit-learn and use it to do the PCA process conveniently and quickly:

In [8]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=1)
x = pca.fit_transform(std_data)
x[:10]
```

Out[8]:
```
array([[-1.70980984],
       [-1.2217608 ],
       [-0.88270861],
       [-0.1753695 ],
       [-1.61057986],
       [-1.05611907],
       [-0.50143101],
       [-0.29388363],
       [ 0.14797534],
       [-1.38746486]])
```

Now, we get a new component x, it's the combination of LON and LAT, we plot it's pair plot with our interested feature SPEED:

In [9]:
```
data['Component'] = x
plot_pair(data, ('Component', 'SPEED'))
```



The situation looks good. Our pair plot of new feature 'Component' and interested feature SPEED looks like the shape of normal distribution.

3. ABC

Since we want to know the positions' effect to the speed of ship. The **prior** is the density of ship's move speed. Our observed data inclued the speed in the different latitude and longitude at the time. **Likelihood** in this case is pretty hard to compute, cause we do not know the speed's effect to the current position, it's continuous and will be pretty hard to construct model also will cost lots of computation sources to do it. Therefore, we need to use Approximate Bayesian Computation to solve this problem approximately. Recently, with the progress of the research in Markov chain Monte Carlo (MCMC) especially the Hamiltonian Monte Carlo (HMC) method, we can get a pretty good simulation result.

We use Stan to implement our model. Stan is a state-of-the-art platform for statistical modeling and high-performance statistical computation. Thousands of users rely on Stan for statistical modeling, data analysis, and prediction in the social, biological, and physical sciences, engineering, and business.

Using Pystan package to call the Stan method in Python. PyStan provides an interface to Stan, a package for Bayesian inference using the No-U-Turn sampler, a variant of Hamiltonian Monte Carlo.

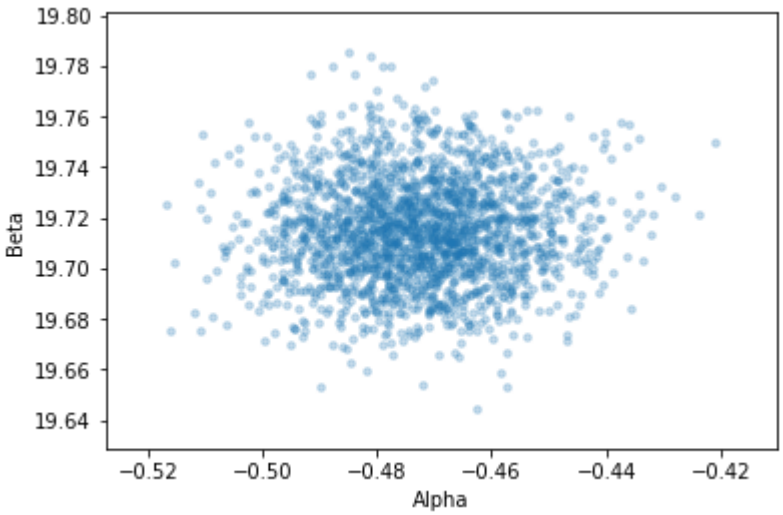
We use four MC chains to sumulate our posterior, and each chain interate for 2000 times, 1000 for warm-up, left 1000 times for sampling. So for each model, we will have 4*1000 = 4000 samples.

In [10]:
```
import pystan
import numpy as np

stan_code = '''
data{
    int<lower=0> N;
    vector[N] x;
    vector[N] y;
}

parameters{
    real alpha;
    real beta;
    real<lower=0> sigma;
}

model{
    y ~ normal(beta + alpha * x, sigma);
}

'''
x = x.reshape(len(x))
y = data['SPEED'].values
model = pystan.StanModel(model_code=stan_code)
fit = model.sampling(data={'x': x[:200], 'y': y[:200], 'N': 200}, seed=16)

print(fit)
```

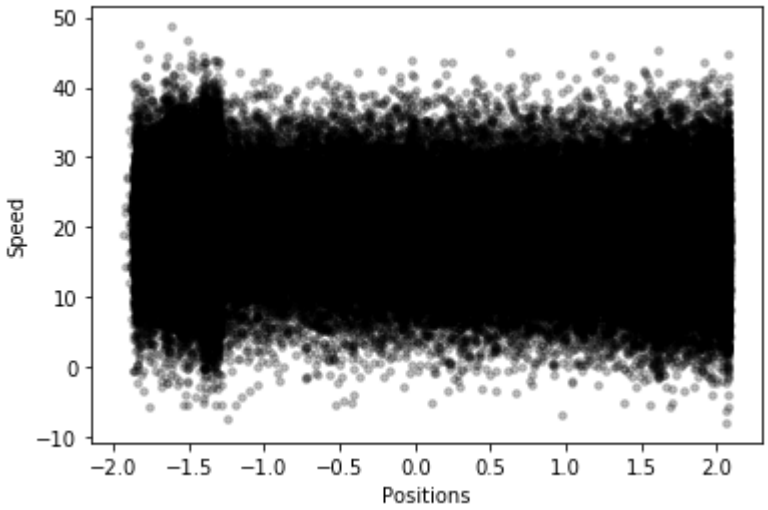
```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_321ef82d03545c8f5
2e0a333c83ee8bd NOW.

Inference for Stan model: anon_model_321ef82d03545c8f52e0a333c83ee8bd.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   R
hat
alpha  -0.75  4.9e-3   0.32  -1.38  -0.96  -0.75  -0.54  -0.12   4280
1.0
beta   19.73  7.6e-3   0.45  18.81  19.45  19.73  20.03  20.61   3482
1.0
sigma   6.37  5.1e-3   0.32   5.79   6.15   6.35   6.58   7.02   3926
1.0
lp__  -467.8    0.03   1.18 -470.7 -468.3 -467.4 -466.9 -466.4   1837
1.0

Samples were drawn using NUTS at Wed Dec  4 13:51:55 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

The values of Rhat and n_eff both show that our sumulation get a convergence and perform well. Now, let us plot the poserior density.

In [11]:
```
import pickle

def read_model(model_name='model_fit.pkl'):
    with open(model_name, "rb") as f:
        data_dict = pickle.load(f)
    fit = data_dict['fit']
    model = data_dict['model']
    return fit, model

fit, model = read_model()
samples = fit.extract(permuted=True)
means = fit.get_posterior_mean()
alpha = round(means[0].mean(), 2)
beta = round(means[1].mean(), 2)
sigma = round(means[2].mean(), 2)
plt.scatter(samples['alpha'][:2000], samples['beta'][:2000], 50, marker=
'.', alpha=0.25)
plt.xlabel('Alpha')
plt.ylabel('Beta')
plt.show()
```



In [12]:
```
def plot_density(alpha, beta, sigma, x):
    y = np.ndarray(len(x))
    for i, k in enumerate(x):
        y[i] = np.random.normal(alpha * x[i] + beta, sigma)
    plt.scatter(x, y, 50, marker='.', alpha=0.25, color='k')
    plt.xlabel('Positions')
    plt.ylabel('Speed')
    plt.show()
    return y

y_sample = plot_density(alpha, beta, sigma, x)
```



In [13]:
```
def plot_density_feature(f):
    plt.figure(figsize=(12, 8))
    plt.tight_layout()
    sns.distplot(f, hist=True, kde=True)
    plt.xlabel('Speed')
    plt.ylabel('Density')
    plt.title('Posterior Density of Speed')
    plt.show()

plot_density_feature(y_sample)
```